University of London
Imperial College of Science, Technology and Medicine
Department of Computing

Enhancing Response Time and Reliability via Speculative Replication and Redundancy

Zhan Qiu

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of the University of London and
the Diploma of Imperial College, February 2017

Declaration of originality

I declare that this thesis was composed by myself, and that the work it presents is my own, except where otherwise stated.

Copyright declaration

The copyright of this thesis rests with the author and is made available under a Creative Commons Attribution Non-Commercial No Derivatives licence. Researchers are free to copy, distribute or transmit the thesis on the condition that they attribute it, that they do not use it for commercial purposes and that they do not alter, transform or build upon it. For any reuse or redistribution, researchers must make clear to others the licence terms of this work.

Abstract

Modern computer systems, such as cloud or server farms, have been widely deployed to provide cost-effective and high-performance services. However, with the increasing scale and complexity, it is not a trivial task to achieve high reliability and consistently low response times. To address these issues, concurrent request replication has emerged as an effective mechanism to achieve these goals. With replication, $r \geq 1$ replicas of each request are spawned simultaneously, and the results of the first k ($1 \leq k \leq r$) replicas to complete are used. Replication can thus handle unpredictable failures and delays due to exceptional conditions, unless they occur to all replicas simultaneously.

The main *risk* of replication is that it may negatively impact latency since it introduces extra load into the system. In this thesis we aim to capture the *trade-off* between these two conflicting effects of replication. We first focus on the $k = 1$ case, i.e., the system replies with the result from whichever replica completes successfully first. We investigate how replication can be used to exploit processing time variability to reduce response times. Next, we investigate the $k = r$ configuration, a case known as fork-join queues, where each arriving job is split into r tasks, each of which is assigned to one of r parallel processors. The key difference with the $k = 1$ case is that the final result can only be delivered once all the tasks have been completed. In the last part, we generalize the analysis to systems that implement erasure encoding, where an object is stored by creating k fragments and storing $r \geq k$ encoded fragments so that the original object can be retrieved from any k of the r fragments.

The models proposed have the advantage of being able to compute the response-time distribution, which is a significant advantage as mean response time guarantees are not sufficient in deadline-driven applications. In addition, these models are able to handle fairly general inter-arrival and service times. Extensive experimental results show that replication can be very effective in keeping the response-time tail short, but these benefits highly depend on the number of replicas, processing-time distribution, as well as on the system load, number of servers, and the statistical characteristics of the arrival process.

Acknowledgements

I can still remember the first day I came to Imperial, but now it is the time to say goodbye. I can not express how much I have learned, grown, and enjoyed working here the last five years. I would like to reflect on the people who have supported and helped me so much throughout this period.

First of all, I would like to express my sincere gratitude to my supervisor Prof. Peter Harrison for his advice, patience, motivation, and immense knowledge. He has been supportive since the days I began working with him as a Master student, giving me the freedom and support to explore new research ideas and collaborations. I would not have the chance to do and finish my Ph.D without his motivation and guidance.

I would also like to thank Dr. Juan Pérez for his incredible support during our collaborations, not only by providing me with great ideas and insightful comments, but also emotionally through the rough road to develop the contributions in this thesis.

Dozens of people have helped and taught me immensely at Imperial, and I would like to thank the following wonderful collaborators for all the fun we have had together: Dr. William Knottenbelt, my second supervisor, for his inspiration and guidance; Dr. Lydia Chen and Dr. Robert Birke, researchers from IBM Research Zurich, for the productive collaboration, and sincere suggestions on my career development; Dr. Naresh Patel for his advice and collaboration on erasure encoding; Dr. Giuliano Casale and Dr. Richard Gibbens, my examiners, for their sincere suggestions. Without their precious support it would not be possible to conduct the research in this thesis.

I would also like to thank Dr. Victoria Lopez Morales, Dr. Daniel Dubois, Dr. Tiberiu Chis, Dr. Pooyan Jamshidi, Dr. Ilya Reshetouski, Dave Wei, Dr. Chen Li, Nicolai Stawinoga, Dr. Gareth Jones, and members of the AESOP for all good time that we have spent together.

Most importantly I thank my parents Qin Liu and Chunhong Qiu for their love, spiritual support and patience. I also want to thank the rest of my family and my friends for their support.

Finally, I want to thank my husband Weikun Wang for everything.

Contents

Abstract **iii**

Acknowledgements **v**

1 Introduction **1**

1.1 Motivation and objectives 1

1.2 Summary and organization 5

1.2.1 Replication in unreliable systems 5

1.2.2 Replication in systems without failures 7

1.2.3 Fork-join queues: 8

1.2.4 Erasure encoding 9

1.2.5 Summary 9

1.3 Publications 10

2 Background Theory **13**

2.1 Laplace transform and generating functions 13

2.1.1 The Laplace transform 13

2.1.2 Generating functions 14

2.1.3 Example 15

2.2 Matrix-analytic methods 18

2.2.1 Quasi-Birth-and-Death processes 18

2.2.2 Finite Quasi-Birth-and-Death processes 19

2.3 Markovian arrival processes 20

2.4 Phase-type distributions 21

2.5 Calculation of the steady state job waiting-time distribution in MAP/PH/1 queues 22

2.5.1 The matrix T 24

2.6 Related Work 27

3 Fault-tolerance and Latency-reduction via Replication: the Distributed Set-up 29

3.1 Reference model 29

3.1.1 The multi-node system 31

3.2 The case with $r = 2$ replicas 32

3.2.1 Conditional request response-time distribution 32

3.2.2 Formulation based on difference between queue-lengths 38

3.3 The case with $r \geq 2$ replicas 40

3.3.1 The waiting-time distribution 41

3.3.2 The service-time distribution 45

3.3.3 Experimental validation 49

3.3.4 Experimental results 50

4 Fault-tolerance and Latency-reduction via Replication: the Centralized Set-up 55

4.1 The case with $r = 2$ replicas . . . 55

4.1.1 The queue-length distribution . . . 56

4.1.2 The service-time distribution . . . 60

4.1.3 The waiting-time distribution . . . 62

4.1.4 Experimental assessment . . . 64

4.2 The case with $r \geq 2$ replicas . . . 66

4.2.1 The waiting-time distribution . . . 66

4.2.2 The service-time distribution . . . 68

4.3 Comparison: distributed vs centralized . . . 70

4.4 The case with parallel jobs . . . 72

4.4.1 Background . . . 73

4.4.2 The job service-time distribution . . . 75

4.4.3 Introducing correlation between replicas . . . 78

4.4.4 Efficient response-time computation . . . 80

4.4.5 Exploiting restricted transitions . . . 81

4.4.6 Evaluation . . . 84

4.4.7 Evaluating the impact of replication . . . 86

4.5 Discussion . . . 92

5 Variability-aware Replication for Latency-reduction in Reliable Systems 94

5.0.1 Motivating example . . . 95

5.1 The distributed set-up: replication without canceling 97

5.1.1 Problem set-up 98

5.1.2 Grouped allocation 99

5.1.3 An approximation for random allocation 104

5.1.4 Experimental results 108

5.2 The centralized set-up: replication without canceling 115

5.2.1 Motivation and system set-up 117

5.2.2 Independent Processing Times 119

5.2.3 Validation 127

5.2.4 Insights regarding replication 130

5.2.5 Correlated processing times 133

5.2.6 Evaluation on MediaWiki 136

5.3 The centralized set-up: replication with canceling 141

5.3.1 Reference model 142

5.3.2 The basic case 143

5.3.3 The general case 147

5.3.4 Experimental assessment 151

5.3.5 Introducing the canceling overhead 154

5.4 Discussion 155

6 Efficient Approximation for Response-Time Tails in Fork-join Queues 157

6.1 Introduction 157

6.2 Existing work 159

6.3 Reference model 160

6.4 The response-time distribution 161

6.4.1 The waiting-time distribution 161

6.4.2 The service-time distribution 163

6.4.3 The response-time distribution 165

6.4.4 Extension to PH services 166

6.4.5 Experimental validation 167

6.5 Approximate analysis 169

6.5.1 Nelson and Tantawi's approximation for mean 170

6.5.2 Efficient approximation for tails (EAT) 171

6.5.3 Asymptotic approximation for Poisson arrivals and exponential services . 173

6.5.4 Asymptotic approximation for MAP arrivals and PH services 176

6.6 Evaluating the EAT approximation 179

6.7 Discussion 180

7 Managing Response-Time Tails by Erasure Encoding 182

7.1 Related work 184

7.2 Model types and definitions 185

7.2.1 Scheduling policies 185

7.3 Canceling model 186

7.3.1 The waiting-time distribution 187

7.3.2 The service-time distribution 190

7.3.3 Resource consumption 192

7.4 Early-canceling model . . . 192
7.4.1 The waiting-time distribution . . . 193
7.4.2 The service-time distribution . . . 195
7.5 Split-merge model . . . 198
7.5.1 The job service-time distribution . . . 198
7.5.2 The job waiting-time distribution . . . 199
7.6 Numerical results . . . 200
7.6.1 Validation . . . 200
7.6.2 Comparing cancelation policies . . . 201
7.6.3 The impact of redundancy level . . . 202
7.6.4 Storage-latency trade-off . . . 204
7.6.5 MAP arrivals . . . 206
7.6.6 Canceling with general processing times . . . 207
7.7 Discussion . . . 208

8 Conclusion **210**

8.1 Summary of Thesis Achievements . . . 210
8.2 Future work . . . 212

Bibliography **213**

A **225**

A.1 Proofs . . . 225
A.1.1 Proof of Proposition 4 . . . 225

A.1.2 Proof of Theorem 3.1 226

A.1.3 Proof of Theorem 5.1 228

A.1.4 Fitting correlated processing times 230

A.2 EM estimation for CHE times 232

List of Tables

1.1 Organization of this thesis 6

2.1 Summary of main notations used in this thesis 14

3.1 Mean and Standard deviation of response times with $\mu = 1$: simulation, and SEM. 39

3.2 Transition rates for matrix S 42

3.3 Transition rates for matrix $A^{(\text{jump})}$ 42

3.4 Transition rates for $\bar{S}_{\text{ser}}$, $\bar{\boldsymbol{S}}_S^*$ and $\bar{\boldsymbol{S}}_F^*$ 45

3.5 Approximation error (%) compared with simulation results 50

4.1 Transition rates for levels $0 \leq k \leq c$ 58

4.2 Transition rates for levels $k \geq c+1$ 60

4.3 Transition rates of $Q_S(k)$, $k \geq \lfloor c/2 \rfloor + 1$ 61

4.4 Transition rates of S and $A^{(\text{jump})}$ 62

4.5 Transition rates of matrix S 66

4.6 Transition rates of matrix $A^{(\text{jump})}$ 67

4.7 Transition rates of S_{ser} 70

4.8 Transition rates for task-level replication 76

4.9 Transition rates for job-level replication 77

4.10 Transition rates for job-level replication with correlated failures 79

4.11 Computation times (sec) for jobs with 50 tasks 85

4.12 Computation times (sec) for jobs with 100 tasks 86

5.1 Transition rates for S, $A^{(\text{jump})}$ and $S_{\text{not-all-busy}}$ 100

5.2 Approximation errors (Grouped) 104

5.3 Approximation errors (Random) 106

5.4 Fitting error 112

5.5 Transition rates for S and $A^{(\text{jump})}$ with PH services 119

5.6 Transition rates for $S_{\text{(all)-(not-all)}}$ and $S_{\text{not-all-busy}}$ 121

5.7 Transition probabilities for $R_{\text{not-all-busy}}$, $R_{\text{all-busy}}$, and R_{ser} 122

5.8 Transition rates for S_{ser} 123

5.9 Transition rates for S and $A^{(\text{jump})}$ - CHE services 134

5.10 Transition rates for $S_{\text{not-all-busy}}$ - CHE services 134

5.11 Transition rates for $S_{\text{(all)-(not-all)}}$ - CHE services 135

5.12 Transition probabilities for $R_{\text{not-all-busy}}$, $R_{\text{all-busy}}$, and R_{ser} - CHE services 135

5.13 Transition rates for S_{ser} - CHE services. Main differences to Table 5.8 136

5.14 Summary of main notations used in this section 143

6.1 Transition rates for S and $A^{(\text{jump})}$ 162

6.2 Transition rates for $S_{\text{not-all-busy}}$ 163

6.3 Transition rates for $S_{\text{(full)-(not-full)}}$ and $S_{\text{not-full}}$ 164

6.4 Transition rates for S and $A^{(\text{jump})}$ with PH services 166

6.5 Transition rates for $S_{\text{not-all-busy}}$ with PH services 166

6.6 Approximation error compared with exact results and simulations 168

7.1 Transition rates for matrices S and $A^{(\text{jump})}$ 189

7.2 Transition rates for $S_{\text{(full)-(not-full)}}$ and $S_{\text{not-full}}$ (Canceling) 190

7.3 Transition rates for matrices $S_{\text{(all),(not-all)}}$ and $S_{\text{not-all-busy}}$ (early-canceling) 193

7.4 Transition probabilities for $R_{\text{all-busy}}$ and $R_{\text{not-all-busy}}$ (early-canceling) 194

7.5 Transition rates for $S_{\text{(full)-(not-full)}}$ (early-canceling) 194

7.6 Transition rates for matrix S_{ser} 198

7.7 Response-time percentiles and the improvement (%) over $r = 4$ 206

List of Figures

1.1 Facebook traces reveal median CPU and memory utilization under 20% (from [AGSS12]). 2

1.2 Reliability 3

2.1 Example to illustrate the age process. 22

3.1 Reference models. A server farm is split in nodes, as in (a). The set-up of each node can be as in (b) or in (c). 30

3.2 Poisson arrivals, 90% baseline reliability 52

3.3 Poisson arrivals, 50% baseline reliability 52

3.4 Poisson arrivals, 10% baseline reliability 53

3.5 MAP arrivals, 90% baseline reliability 53

3.6 Improvement on R_p (Poisson arrivals, reliability: 90%) 54

3.7 Improvement on R_p (Poisson arrivals, reliability: 50%) 54

4.1 Reference model 56

4.2 The failure rate effect 64

4.3 The IAT process effect 64

4.4 The number of servers 65

4.5 Utilization threshold 65

4.6 CCDFs of response times with the distributed set-up queues and with a centralized set-up (Poisson arrivals, $r = 2$, utilization:0.3) 68

4.7 CCDFs of response times with the distributed set-up queues and with a centralized set-up (Poisson arrivals, $r = 3$, utilization:0.3) 68

4.8 Improvement on R_{95} of the centralized set-up over the distributed set-up 71

4.9 Split-merge queue 73

4.10 Reference Model 73

4.11 Reference job-level and task-level models 74

4.12 Reliability under different replication modes 88

4.13 The effect of the correlation - baseline reliability: 90% 89

4.14 The effect of the correlation - baseline reliability: 50% 90

4.15 R_{95} achieved with different baseline utilizations 90

5.1 Improvement on percentiles 95

5.2 Replication model 98

5.3 Response time mean and percentiles under $r = 1, 2$ 108

5.4 Utilization threshold (random) 109

5.5 $R_r(99)$ improvement (random) 109

5.6 Utilization threshold (grouped) 110

5.7 $R_r(99)$ improvement (grouped) 110

5.8 The impact of Variability 111

5.9 The impact of Skewness 112

5.10 Log C services 112

5.11 Percentile improvement in the case study 114

5.12 Improvement with $r > 2$ replicas 116

5.13 Dense vs. sparse setup: response-time CCDF 117

5.14 Response times under replication. 117

5.15 Distribution of processing times of two replicas from the same request. 118

5.16 Reference model. 118

5.17 Experimental validation with the MATLAB benchmark on response-time percentiles. Each experiment is labeled by a tuple of (arrival process, processing-time distribution, number of servers). B and M denote results obtained from the benchmark and the model, respectively. 128

5.18 Percentile improvement. 130

5.19 Threshold utilization. 131

5.20 Experimental validation with the MediaWiki. 138

5.21 Optimal replication factor: measurement vs. model. 138

5.22 Improvement using optimal factor r^*. 139

5.23 Prediction errors using different fitting methods. 139

5.24 Prediction errors using $\lambda = 5$. 140

5.25 The basic and the general cases 142

5.26 Service time PDF and response-time CCDF 152

5.27 Response times under different baseline utilizations and number of replicas . . . 153

5.28 $R_{99.99}$s under different overhead with Exp services 154

6.1 Fork-join queue 160

6.2 Errors(%) of approximation 176

6.3 Errors(%) of approximation for $R_r(95)$ . . . 179

6.4 Errors(%) of approximation when $U = 0.5$ . . . 180

7.1 Canceling example . . . 185

7.2 Early-canceling example . . . 185

7.3 Approximation error w.r.t. simulation – $C = 10$ . . . 200

7.4 Comparison of three policies – $(r, k) = (9, 6)$, $C = 10$. . . . 201

7.5 Comparison of the redundancy level $r - k$ (Canceling case, $k = 6$, $C = 10$). . . . 201

7.6 Improvement on response time's 99^{th} percentile for canceling over early-canceling ($C = 10$) . . . 202

7.7 Trade-off between storage space r/k (corresponding to decreasing k horizontally) and 99th percentile of response time ($r = 9$, $k = \{1, \ldots, 9\}$, $C = 10$). . . . 203

7.8 Response-time PDF under Possion and MAP arrivals; (r,k)=(6,4), utilization 0.5, $C = 10$. . . . 206

7.9 Response-time PDF ($k = 4$, MAP arrivals, utilization 0.5, $r = 4$, 5, 6, 7, $C = 10$). 206

7.10 The split-merge model provides an upper bound on the canceling model – (r,k)=(9,6), $C = 10$. . . . 207

Chapter 1

Introduction

1.1 Motivation and objectives

Fueled by the ever-growing demand for computation-intensive and massive-data operations, modern computer systems, such as cloud, data centers, or server farms, have been widely deployed to provide cost-effective and high-performance services [Mit13, KB03, KMR08]. They are usually composed of a set of local or geographically distributed, inexpensive processing elements, providing computing services to multiple users. Besides low-cost access, two major demands of users are to consistently achieve high reliability and low latency. However, these systems, as any computing system, are subject to failures, and their overall reliability deteriorates with increasing scale and complexity [KMR08], degrading the offered quality of service (QoS). In this thesis we consider failures at the application level, also referred to as *request failures*, caused for instance by deadlocks, errors in the input data, communication errors [ÖV96], timeouts of resources with constrained availability, or application outputs exceeding latency requirements. To address this problem, various fault-tolerance mechanisms have been developed, among which, checkpoint/restart [FSLI+11] has been the most prevailing. In particular, in the checkpoint/restart approach, a failed request is re-executed, either from the beginning or from the last checkpoint. Although effective to handle failures, a major concern with this approach is the additional delay it introduces, degrading the application

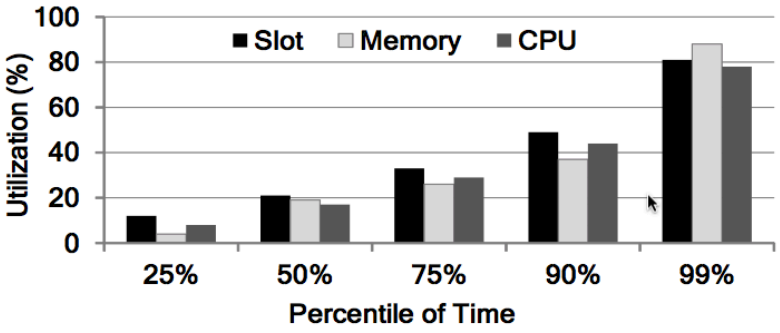


Figure 1.1: Facebook traces reveal median CPU and memory utilization under 20% (from [AGSS12]).

QoS, which is a major issue for deadline-driven applications [VMGS12, Bru09]. Moreover, this approach has been found to impose large overheads and storage requirements to read and write checkpoints [FSLI+11, OAT+07].

Besides the reliability issue, a key challenge in managing computing systems is to consistently achieve low response times, or latency, especially to keep the tail of the response-time distribution short. For example, it has been observed that virtualization in cloud systems can harm performance owing to co-located virtual machines competing for CPU, memory bandwidth or other resources. Experiments of Google show that a system where each request typically responds in 1ms, has a 99^{th} percentile latency of 10ms [DB13]. Thus, a non-negligible fraction of the processed requests face latencies that are very far from the average case. High latency can be caused for a number of reasons, including contention for shared resources, queueing, imbalance in request workload or hardware problems [DB13, AKG+10]. This performance degradation leads to an increased variability in processing times, which impacts the application response times, particularly affecting those users that face the longest response times. Keeping latency low, especially to keep the tail of the latency distribution short, is critical since even a small amount of delay may significantly degrade the perceived QoS of a system. For instance, the traffic and revenue of Google searches has been observed to decrease by 20% when introducing a 500ms delay [Lin06]. However, it is challenging for service providers to keep variability in latency under control due to the scale and complexity of computing systems nowadays. Recent efforts [DB13, AKG+10, DG08] have proposed approaches by relying on standard techniques including differentiating service classes, managing background activities and reissuing requests.

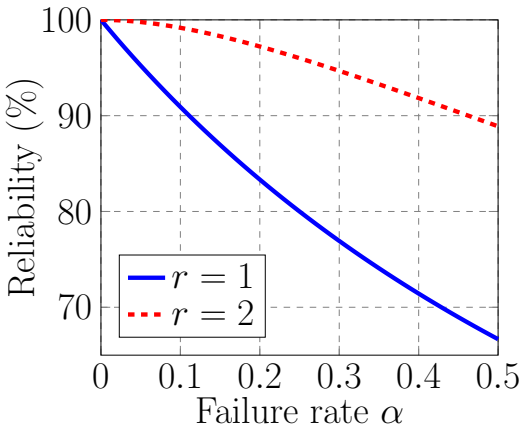


Figure 1.2: Reliability

However, as most systems have shared computational resources, and with the increasing of the scale and complexity of modern computing systems, it is expensive or even infeasible to eliminate all latency variability using existing engineering techniques [DB13].

To address these issues, *concurrent or speculative request replication* has been proposed [DB13]: $r \geq 1$ replicas of each request are spawned simultaneously, and the results of the first k replicas ($1 \leq k \leq r$) that complete are used. This approach has been recently considered in [DB13, AGSS12, SLR16, VMGS12] for the special case $k = 1$, i.e., the request latency is determined by the minimum latency among all its replicas; and in [PGK88, CFG+05] with $k \geq 1$ for deadline-driven and mission-critical systems to improve both latency and reliability.

Heavy concurrent replication is appealing given the common observation that most clusters are highly underutilized, with the average utilization of major data center servers being around 18% [Sny10]. For example, traces from the Facebook cluster, as shown in Figure 1.1, reveal that CPU and memory utilizations have a median of less than 20% [AGSS12]. Further, as it is sufficient if only some of the replicas succeed, concurrent replication can handle unpredictable failures, unless they occur to all replicas simultaneously.

To illustrate the effectiveness of replication on improving job reliability, we consider a simple set-up where both service and failure times are exponentially distributed, a common assumption in reliability engineering [ZL09, WSJ07]. In particular, we assume that servers process incoming requests with rate μ, and requests in service are subject to failures, with failure rate α.

The reliability, measured as the probability that a request completes service successfully, thus depends only on the service and failure rates, being given by $\mu/(\mu+\alpha)$ and $1-(\alpha/(\mu+\alpha))^r$ for the systems without replication, and with r replicas (including the original copy), respectively. Assuming $\mu = 1$, Figure 1.2 compares the reliability achieved under failure rates α between 0.1 and 0.5, for systems without replication ($r = 1$), and with $r = 2$, showing a significant improvement in reliability when introducing replication. For instance, even under a large failure rate of 0.5, the reliability improves from 66.67% to 88.89% thanks to the added replica, confirming the effectiveness of replication to improve reliability.

In addition to the benefit on the reliability, this approach has the potential to reduce the request latency. The motivation behind this approach comes from the observed high variability in processing times from the server side, as servers might be temporarily slow down due to factors such as current loads, network congestion, resource availability, background interference or hardware problems. Concurrent replication can thus handle unpredictable delays due to exceptional conditions, unless they occur to all replicas simultaneously. Recent studies have shown that the source of latency is often due to external interference, rather than inherent to the requests. This interference causes the request latency to be unpredictably long. For example, a web query can take up to a factor of 28 times longer due to background activities [XBNJ13]. Replication can potentially reduce both the mean and the tail of the latency distribution, as the overall latency of a request becomes the delay of the first k across all its replicas. Sending redundant requests enables a job to receive the minimum service time across servers, thus being effective in handling exceptional delays unless they occur in all replicas simultaneously. In addition, in the distributed set-up where each server keeps its own queue, replication provides further improvements because it can exploit the variability of the queueing times. For instance, replication of DNS queries can mask delays caused by cache misses, network congestion, or packet losses [VGM+13].

Although being effective to handle both failures and unpredictable delays, the main *risk* of replication is that it may negatively impact latency since it introduces extra load into the system, which may lead to excessive delays. In this thesis we aim to capture the *trade-off* between these two conflicting effects of replication, and characterize the scenarios under which replica-

tion improves latency tails, in addition to improving reliability. We investigate how speculative request replication can be used to exploit processing time variability to reduce response times, considering not only mean values but also the *tail* of the response-time distribution. Evaluating the effect of replication is, however, non-trivial, for a number of factors, such as system load and server pool size, will be shown in this thesis to affect the response time.

1.2 Summary and organization

Determining the job response-time distribution, especially its tails, requires expensive experiments or time-consuming simulations. However, the models proposed in this thesis allow rapid evaluation of different scenarios, providing a theoretical guidance on when to adopt replication, and how many replicas to adopt for real applications. This section summarizes the key contributions of the three parts of this thesis, where we describe in Table 1.1 the organization of chapters within these parts. In the first part of this thesis (Chapters 3, 4 and 5), we consider first-to-finish request replication to improve reliability and reduce latency: $r-1$ extra replicas are adopted for each arriving request, i.e., a total of r replicas of a request are submitted to the system. In particular, $r = 1$ represents the case where no replication is adopted. The system then replies with the result from whichever replica completes successfully first, i.e., $k = 1$. The outstanding replicas of the same request, can either continue service, or be canceled immediately to reduce unnecessary workload.

1.2.1 Replication in unreliable systems

In Chapters 3 and 4 we focus on systems where requests are subject to failures, assuming that the server is not affected by the failure, and continues serving the next request at the front of the queue. We consider both the distributed and centralized set-ups, and develop stochastic models to analyze the request latency distributions that enable the comparison of the performance of these two set-ups.

Table 1.1: Organization of this thesis

k	**Common set-up**	**Chapter/Section**	**Details**
$k = 1$	Request failures, with canceling	Chapter 3	Distributed set-up
		Chapter 4	Centralized set-up
	Reliable systems	Section 5.1	Distributed set-up without canceling
		Section 5.2	Centralized set-up without canceling
		Section 5.3	Centralized set-up with canceling
$k = r$	Fork-join queue	Chapter 6	Distributed set-up
$1 \leq k \leq r$	Erasure encoding	Chapter 7	Distributed set-up

In the distributed set-up, as described in Chapter 3, we consider a server farm consisting of a number of distributed, homogeneous, and independent servers, where each server has its individual queue, and processes replicas with first-come first-served (FCFS) scheduling. We start with the simple case with $r = 2$ in Section 3.2, where we consider exponential processing times and a Poisson arrival process. We then develop an approximate stochastic model in Section 3.3 for the more general set-up with Markovian arrivals and $r \geq 2$, which allows the determination of the request response-time distribution. Although approximate, validation results indicate that our model is accurate for not only the mean response time but also its percentiles, a significant advantage since mean response time guarantees are not sufficient in many situations, in particular for deadline-driven applications. An alternative set-up is to have a centralized queue in front of all the servers, as in Chapter 4, where all the replicas of the incoming requests form a single queue in the order of arrival and join the next server that becomes available with FCFS scheduling. In Sections 4.1 and 4.2 we propose a stochastic model to study how the replication approach affects the offered request response-time distribution.

In terms of response times, the centralized set-up must achieve lower response times than its distributed counterpart, as in the latter case it is possible for some servers to have non-zero waiting lines, while other servers remain idle. This is not possible in the centralized set-up, as all servers must be busy if there is any job waiting to start service. However, most large server farms dispatch incoming request to servers immediately, without holding a central queue. This set-up provides more flexibility since there is no need to set up and manage a central dispatcher that keeps track of the servers state, making it easier to increase or decrease the number of servers as the load rises or falls. A major point of our investigation is to assess the magnitude of the difference in response time between the two set-ups, to see if and when the distributed set-up

suffers an excessive performance penalty. From extensive experimental results in Section 4.3, we show that the distributed set-up offers response times close to those of the centralized system, especially when the request reliability is medium or high.

We then extend our analysis to consider parallel jobs that are composed of multiple tasks in Section 4.4, where the job response time is determined by the last task to finish. To understand the impact of speculative replication on the reliability and response time of parallel jobs, we propose a stochastic model to determine the response-time distribution, considering both task-level and job-level replication.

1.2.2 Replication in systems without failures

We then focus on the impact that speculative request replication has on reducing the request latency in Chapter 5, considering systems without failures. Here we relax the exponential service time assumption to general phase-type request processing times [LR99], which are necessary to capture the variability observed in the server side. We first focus on the *distributed* set-up in Section 5.1, then consider its counterpart, the centralized set-up, in Sections 5.2 and 5.3. Here we consider two cases: replication without canceling in Section 5.2, and we introduce canceling in Section 5.3. Whereas canceling has obvious benefits in limiting the additional load introduced by replication, it can be expensive or unfeasible to implement. Results show that, the potential of replication to reduce latency, depends on a number of factors, including the arrival and service processes, the canceling policy, the utilization, and the number of replicas. Service time variability plays a key role when evaluating the effect of replication on latency, where highly variable service times provide larger gains, and offer more chances to benefit from replication. Further, we observe that the improvement on latency is not uniform across different percentiles, and the load threshold under which replication helps to reduce latency largely depends on the metric targeted. Explicitly considering the response-time distribution, via the targeted percentiles, is therefore fundamental to evaluate a replication strategy.

1.2.3 Fork-join queues:

The above cases focus on the first-to-finish $k = 1$ case, where the request benefits from the high variability in processing times from the server side as its latency becomes the minimum delay across all its r replicas. On the contrary, in many parallel systems, after a step where many tasks are executed in parallel by a set of resources, the final product (job) can only be delivered once all the tasks have been completed, thus introducing a synchronization point. Parallelism can be found in many different computer systems, such as query processing in distributed databases, parallel computing environments, distributed stream-processing systems, and parallel disk access to improve I/O throughput in storage systems, amongst others. Modern computer and manufacturing systems rely heavily on parallelism to satisfy their performance requirements, exploiting large resource pools to perform tasks that would otherwise be uneconomical or even infeasible. For instance, the Google search engine relies on a high degree of parallelism to return a huge volume of answers in less than a second [Tho15]. *Fork-join queues* are a popular abstraction for these kinds of operations, as they are able to capture the parallel execution and synchronization aspects precisely.

A key performance metric in a fork-join queue is the job response time, which is determined by the last task to finish. Its computation is notoriously difficult due to the synchronization step. In fact, exact solutions for the *mean* response time are only available for fork-join queues with $r = 2$ nodes. In Chapter 6, we propose an *Efficient Approximation for Response-Time Tails (EAT)* of fork-join jobs, which relies on properties of order statistics and extreme values. EAT has the advantage of being efficient and scalable, requiring only the analytical results for a single-node and 2-node fork-join queues, which we obtain by the matrix-analytic approach. Experimental results show that EAT yields good fits for the tails, even in very large cases with general processing and inter-arrival times. The scalability and accuracy of EAT make it a good candidate to support scheduling, resource provisioning and optimization decisions.

1.2.4 Erasure encoding

Storage costs are increasingly taking a higher proportion of corporate operating and capital budgets as data growth continues unabated. As a result, ways are being sought to reduce costs by simplifying and consolidating operations while maintaining service level objectives (SLOs) for performance, data management and data protection. With *erasure encoding*, an object is stored by creating k fragments and storing $r \geq k$ encoded fragments so that the original object can be retrieved from any k of the r fragments (whence $r - k$ failures can be tolerated). These can be described as (r, k) *partial-join queues* where the special case with $k = 1$ is simple replication, and $r = k + 1, r = k + 2$ or $r = k + 3$ can represent single, dual or triple parity scenarios. In Chapter 7, we generalize the analysis for systems that implement erasure encoding, where the key issue addressed is to deal with general order statistics of request response times, rather than just the special cases of the minimum and maximum, corresponding to first-to-finish replication, and fork-join queues, respectively. We consider two encoding policies based on when redundant tasks are canceled: (a) "*Canceling*", in which $r - k$ associated tasks terminate their service immediately after the first k complete; (b) "*Early canceling*", in which $r - k$ associated tasks leave the system when any k of the tasks have *started* service. Experimental results show that both canceling policies significantly reduce the high quantiles of job response time, where canceling achieves the lowest response times as it benefits from the property of resource diversity, whilst also limiting the extra load introduced.

1.2.5 Summary

Summarizing, we consider in this thesis the *speculative request replication* approach, where $r \geq 1$ replicas of each request are spawned simultaneously, and the results of the first k replicas $(1 \leq k \leq r)$ that complete are used. We introduce stochastic models to evaluate how replication affects the SLOs in various systems, especially the overall reliability and response-time percentiles. In particular, we consider three different cases: $k = 1$ first-to-finish replication; $k = r$ fork-join queues; and $k \leq r$ erasure encoding. The proposed models allow us to characterize the conditions under which replication effectively reduces the response times.

1.3 Publications

International conference and workshop papers

[RPQ+17] R. Birke, Z. Qiu, J. F. Pérez, M. Bjoerkqvist, L. Chen. **Power of Redundancy: Designing Partial Replication for Multi-tier Applications.** In: *IEEE International Conference on Computer Communications (INFOCOM), 2017.*

This paper investigates the effectiveness of replication for distributed multi-tier systems, where interference from neighboring workloads is entangled with complex tier inter-dependency. My contribution to this paper is the help in designng and implementing the models, and the evaluation of the experiment results.

[QPH16b] Z. Qiu, J. F. Pérez, P. G. Harrison. **Variability-aware Request Replication for Latency Curtailment.** In: *IEEE International Conference on Computer Communications (INFOCOM), 2016.*

This paper investigates how request replication can be used to exploit processing time variability to reduce response times in reliable systems, focusing on the distributed set-up. My contribution to this paper is the development and implementation of the stochastic model, and the system simulation.

[QPH16a] Z. Qiu, J. F. Pérez, P. G. Harrison. **Tackling Latency via Replication in Distributed Systems.** In: *ACM/SPEC International Conference on Performance Engineering (ICPE), 2016.*

This paper develops an approximate stochastic model to determine the response-time distribution in an unreliable system with distributed queues, and compares its performance against its centralized counterpart. I contributed the majority of the paper and the implementation and the evaluation of the proposed methods.

[QP15a] Z. Qiu, J. F. Pérez. **Enhancing Reliability and Response Times via Replication in Computing Clusters.** In: *IEEE International Conference on Computer Communications (INFOCOM), 2015.*

This paper proposes a stochastic model to study how concurrent replication with canceling affects the response times in an unreliable system with the centralized set-up. My contribution to this paper is the development and implementation of the model, and the evaluation of the resource provisioning decisions.

[QP15c] Z. Qiu, J. F. Pérez. **Evaluating the Effectiveness of Replication for Tail-tolerance.** In: *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2015.*

This paper evaluates the impact of replication with canceling has for reliable computing systems with the centralized set-up. I contributed the majority of the paper, and the development and implementation of the model. I also evaluated the optimal number replication

[QP14] Z. Qiu, J. F. Pérez. **Assessing the Impact of Concurrent Replication with Canceling in Parallel Jobs.** In: *IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014.*

This paper explores the impact of replication has on the reliability and mean response time of parallel jobs. My contribution is the development and implementation of the model for both job- and task-level replication, and the evaluation of the effectiveness of replication.

[HQ13] P. G. Harrison, Z. Qiu. **Performance Enhancement by Means of Task Replication.** In: *European Workshop on Performance Engineering (EPEW), 2013.*

This paper considers a pair of distributed queues where requests may encounter failures during service, utilizing the generating function and spectral expansion methods to determine the job response-time distribution. My contribution to this paper is the derivation of the evolved equations, and the implementation of the model and the simulation.

Journal papers

[QP16] Z. Qiu, J. F. Pérez. **Evaluating Replication for Parallel Jobs: an Efficient Approach**. In: *IEEE Transactions on Parallel and Distributed Systems (TPDS), 2015.*

This is an extension of [QP14], which proposes an efficient method to compute the response-

time distribution of jobs with a large number of servers. It also introduced a novel way to model correlated failures among the replicas of a job or task. I contributed to this paper by developing and implementing the stochastic model, and developing the way to model correlated failures.

[QPH15] Z. Qiu, J. F. Pérez, P. G. Harrison. **Beyond the Mean in Fork-Join Queues: Efficient Approximation for Response-Time Tails.** In: *International Symposium on Computer Performance, Modeling, Measurements and Evaluation 2015 (IFIP Performance), 2015.*

This paper introduces a stochastic model to evaluate the response-time distribution for homogeneous fork-join queues. To tackle large-scale scenarios with high loads, an efficient algorithm using matrix-analytic methods is developed. I contributed the majority of the paper by developing and implementing the stochastic model, and the approximation approach, as well as the implementation of the system simulation.

Chapter 2

Background Theory

This chapter provides an overview of the relevant background theory that we use and extend in this thesis, and a summary of existing work in modeling replication. Table 2.1 summarizes the notation that will be used throughout.

2.1 Laplace transform and generating functions

In Section 3.2 we make use of the generating function method to determine the Laplace transform of the request response-time probability distribution. This chapter thus starts with a brief overview of Laplace transform and generating functions, and provides an example to show how to combine these two to obtain the request response-time distribution.

2.1.1 The Laplace transform

For a function f defined on the non-negative reals, we define the Laplace transform, $f^*(s)$, by

$$f^*(s) = (\mathcal{L}f)(s) = \int_0^\infty e^{-st} f(t) dt,$$

Table 2.1: Summary of main notations used in this thesis

Parameter	Definition
$\text{MAP}(D_0, D_1)$	Job arrival process.
λ	Mean arrival rate.
$\text{PH}(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$	Job waiting-time PH distribution.
$\text{PH}(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$	Job service-time PH distribution.
$\text{PH}(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$	Job response-time PH distribution.
$A^{(\text{jump})}, S$	Transition rates with or without a service completion.
γ	Probability that a job has to wait upon arrival.
$\mathbf{1}$	A column vector with unit elements.
$\boldsymbol{e}_s$	A zero vector with a 1 in its s-th element.
I_n	Identity matrix of size n.
SCV	Squared coefficient of variation.
η_0	Number of service completions in a busy period.
η_1	Number of arrivals in a not-busy period.

for real values $s \geq 0$. The Laplace transform is an expectation $f^*(s) = E[e^{-sX}]$ for non-negative random variable X with density f.

2.1.2 Generating functions

A generating function of a random variable is a parametrized expectation defined over that random variable's probability distribution, density or mass function. Let p_x be the mass function of a random variable X whose values are the non-negative integers, and $p_i = p_x(i)$ if $p_x(i) > 0$, $p_i = 0$ otherwise, then the probability generating function G of X is defined by [HP92]

$$G_X(z) = E[z^X] = \sum_{i=0}^{\infty} p_i z^i = p_0 + p_1 z + p_2 z^2 + \cdots + p_k z^k + \ldots.$$

Since $\sum_{i=0}^{\infty} p_i = 1$, it follows that $G_X(1) = 1$. Expectations and higher moments of a random variable may be obtained directly from the probability generating function by differentiation and evaluation at $z = 1$. For example, we have

$$E[X] = \sum_{i=0}^{\infty} i p_i = \lim_{z \to 1} \left(\sum_{i=0}^{\infty} i p_i z^{i-1} \right) = \lim_{z \to 1} G'_X(z) = G'_X(1).$$

The probability distribution of a discrete random variable X is uniquely determined by its generating function, i.e., if two discrete random variables X and Y have the same probability generating function, they must also have the same probability distribution and probability mass function.

2.1.3 Example

As an example, we consider a simple M/M/1 queue, i.e., a single server queue with FCFS scheduling policy, with Poisson arrival process of rate λ and exponentially distributed service process of rate μ. In conventional queueing terminology, we focus on a tagged request, and seek the probability that its response time R does not exceed a positive real value t, conditioned on the numbers of requests N ahead of it at its arrival instant. For integers $n \geq 0$, we define

$$F_n(t) = \Pr(R \leq t \mid N = n).$$

Assuming that the tagged request arrives at time 0, let $N(t)$ be the numbers of requests ahead of the tagged request at time t. For $t > 0$, consider the interval $(0, t + h]$ and the initial infinitesimal sub-interval $(0, h]$, we have:

$$\begin{aligned} F_n(t+h) &= \Pr(R \leq t+h \mid N(0) = n) \\ &= \sum_{i=0}^{\infty} (\Pr(R \leq t+h \mid N(0) = n, N(h) = i) \times \Pr(N(h) = i \mid N(0) = n)) \end{aligned}$$

by the law of total probability. The terms $\Pr(N(h) = i \mid N(0) = n)$ correspond to the possible events that can occur during the interval $(0, h]$ for $h > 0$, which we take to be infinitesimal.

Notice that we do not keep track of the arrivals that arrive later than the tagged request, as their arrival will not influence the response time of the tagged request due to the FCFS policy. Thus when $n > 0$, the only possible events in the interval $(0, h]$ is a departure of the request in service, with probability $\mu h + o(h)$ if $i = n - 1$, by the memoryless property of the exponential

random variable. Therefore, $F_n(t+h)$ satisfies:

$$\begin{aligned} F_n(t+h) = \mu h \Pr(R \leq t+h \mid N(0) = n, N(h) = n-1) \\ + (1-\mu h) \Pr(R \leq t+h \mid N(0) = N(h) = n) + o(h) \end{aligned} \tag{2.1}$$

By the Markov property applied at time h and the residual life property of exponential random variables, the remaining response time of the tagged request after time h, $R_h = R - h$ say, is distributed as the full response time of a hypothetical arrival at time h conditioned on there being N requests ahead [HP93]. For instance, $\Pr(R \leq t+h \mid N(0) = N(h) = n)$ simplifies to

$$\Pr(R_h \leq t \mid N(h) = n) = \Pr(R \leq t \mid N(0) = n) = F_n(t)$$

Thus Eq. (2.1) becomes

$$F_n(t+h) = \mu h F_{n-1}(t) + (1-\mu h) F_n(t) + o(h)$$

Rearranging, dividing by h, taking the limit $h \to 0$ and omitting the argument t for brevity, we obtain

$$f_n = \mu F_{n-1} - \mu F_n,$$

where $f_n(t)$ is the derivative of $F_n(t)$ with respect to t. Let $F_n^*(s) = (\mathcal{L}F_n)(s)$ be the Laplace transform of $F_n(t)$, abbreviated as $L_n(s)$. Taking the Laplace transform of the above equation, noting that $(\mathcal{L}f_n)(s) = sL_n(s)$, we obtain

$$(\mu + s) L_n = \mu L_{n-1}$$

This recurrence relation can then be solved by introducing the generating function $G(x) = \sum_{n=0}^{\infty} L_n x^n$. Multiplying throughout by x^n and summing from $n = 1$ to ∞ then gives

$$(\mu + s)(G(x) - L_0) = \mu x G(x). \tag{2.2}$$

To obtain L_0, we consider the case with $n = 0$, where we have

$$F_0(t + h) = \mu h \cdot 1 + (1 - \mu h)F_0(t) + o(h).$$

The term $\mu h \cdot 1$ corresponds to the tagged request completing service. Proceeding as above, L_0 can be obtained by considering that

$$(\mu + s)L_0 = \mu/s.$$

Combing with Eq. (2.2), we have

$$G(x) = \frac{\mu}{s(s + \mu - \mu x)}.$$

Under FCFS discipline, the unconditional probability distribution of response time at equilibrium, $R(t)$, is given by $R(t) = Pr(R \leq t) = \sum_{n \geq 0} \pi_n F_n(t)$, which has Laplace transform

$$R^*(s) = (\mathcal{L}R)(s) = \sum_{n \geq 0} \pi_n L_n(s),$$

where π_n is the steady state probability the queue length is n, assuming the queue is stable. As the queue-length in the M/M/1 queue is geometric with parameter $1-\rho$, we have $\pi_n = (1-\rho)\rho^n$, where $\rho = \lambda/\mu < 1$ is the utilization of the M/M/1 queue. Then the Laplace transform of $R(t)$ is

$$R^*(s) = (1 - \rho)G(\rho) = \frac{\mu - \lambda}{s(s + \mu - \lambda)}.$$

Notice that the above solution is probably the longest one to find the request response time in a M/M/1 queue. It serves in this thesis as an example to demonstrate the method of using generating functions to find the request response-time distribution.

2.2 Matrix-analytic methods

Matrix-analytic methods [Neu81] have been widely used as a technique to solve two-dimensional Markov chains where the chains grow unboundedly in one dimension (increasing number of jobs), yet have only a finite number of states in the other dimension with a repeating structure [Neu81]. Matrix-analytic methods are numerical methods, and do not provide a closed-form symbolic solution. In this thesis, we rely on these methods to find the request response-time distribution for systems with replications with different set-ups. In the following, we start with the introduction of Quasi-Birth-and-Death (QBD) processes and the Finite QBD processes.

2.2.1 Quasi-Birth-and-Death processes

A continuous-time Quasi-Birth-and-Death (QBD) process [LR99] is a two-dimensional Markov chain (MC) $\{(N(t), J(t)), t \geq 0\}$. $N(t)$ is called the *level* variable, which takes values on $\mathbb{N}$, and $J(t)$ is called the *phase* variable, and takes values on the set $\{1, 2, \ldots, m_0\}$ when the level is 0, and on $\{1, 2, \ldots, m\}$ for other levels. The one-step transitions are independent of the level n for $n > 1$, and restricted to states in levels $n' = n - 1, n, n + 1$. The QBD infinitesimal generator has the form

$$Q = \begin{bmatrix} B_1 & B_2 & 0 & 0 & \cdots \\ B_0 & A_1 & A_2 & 0 & \cdots \\ 0 & A_0 & A_1 & A_2 & \cdots \\ 0 & 0 & A_0 & A_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{2.3}$$

The matrices A_0 and A_2 are non-negative and carry the transition rates from level $n > 1$ to levels $n - 1$ and $n + 1$, respectively. The matrices A_1 and B_1 have non-negative off-diagonal elements, and strictly negative diagonals, holding the transition rates within level n, for $n > 0$ and $n = 0$, respectively. The matrices B_2 and B_0 are non-negative, and hold the transition rates from level 0 to level 1, and from level 1 to level 0, respectively.

For a QBD to be positive recurrent, it requires to hold the following condition that

$$\varrho = \frac{\boldsymbol{\alpha}_A A_2 \mathbf{1}}{\boldsymbol{\alpha}_A A_0 \mathbf{1}} < 1. \tag{2.4}$$

Here ϱ is the drift of the MC, and $\boldsymbol{\alpha}_A$ is the unique solution of the system $\boldsymbol{\alpha}_A A = \mathbf{0}, \boldsymbol{\alpha}_A \mathbf{1} = 1$, with $A = A_0 + A_1 + A_2$. Let $\boldsymbol{\pi}$ be the steady-state distribution of the MC with generator Q, which we partition conformally with Q, i.e., $\boldsymbol{\pi} = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \dots)$. The stationary distribution vector $\boldsymbol{\pi}$ exists if the QBD MC is positive recurrent, and if it exists, can be found as $\boldsymbol{\pi}_i = \boldsymbol{\pi}_1 R^i$, for $i > 1$, where the matrix R is given by

$$R = A_2(-A_1 - A_2 G)^{-1},$$

and G is the minimal non-negative solution of the quadratic matrix equation [LR99]

$$A_0 + A_1 G + A_2 G^2 = 0. \tag{2.5}$$

Many iterative algorithms have been developed to solve Eq. (2.5), including Functional Iterations (FI), Logarithmic Reduction, and Cyclic Reduction (CR) [BMSVH06, LR99].

2.2.2 Finite Quasi-Birth-and-Death processes

Unlike the level-independent QBD, the transitions of the finite QBD processes are explicitly dependent on the level, where the state-space of the level variable $N(t)$ is finite [GJL84]. The finite QBD processes can be defined similarly using the pair of variables $\{(N(t), J(t)|t \geq 0\}$ on the state space $\{(k, j)|0 \leq k \leq N, j \in \{1, 2, \dots, m_k\}\}$, where m_k is the number of phases in level k.

The corresponding block-tridiagonal infinitesimal generator Q_{finite} has the following structure

$$Q_{\text{finite}} = \begin{bmatrix} A_1^{(0)} & A_2^{(0)} & 0 & 0 & \cdots & 0 \\ A_0^{(1)} & A_1^{(1)} & A_2^{(1)} & 0 & \cdots & 0 \\ 0 & A_0^{(2)} & A_1^{(2)} & A_2^{(2)} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & A_0^{(N)} & A_1^{(N)} \end{bmatrix},$$

and has row sums equaling zero and is comprised of level-dependent blocks, where $A_1^{(k)}$ $(0 \leq k \leq N)$ are square matrices, with order m_k and matrices $A_2^{(k)}$ $(0 \leq k \leq N-1)$ and $A_0^{(k)}$ $(1 \leq k \leq N)$ are rectangular. The steady-state distribution $\boldsymbol{\pi}$ of the QBD, can be found following [GJL84, Algorithm B].

2.3 Markovian arrival processes

In this thesis, request arrivals at the system are modelled as a Markovian arrival process (MAP). MAPs are widely used to represent arrival streams with general and correlated inter-arrival times (IATs) [LR99]. The continuous-time MAP [LR99] is a marked MC with generator $D = D_0 + D_1$, where the transition rates not associated with arrivals are held in D_0, while the rates that trigger new arrivals are kept in D_1. The diagonal entries of D_0 hold the total exit rate in each state, such that $(D_0 + D_1)\mathbf{1} = 0$. We denote this process as MAP(m_a, D_0, D_1), where m_a is the number of states, or arrival phases, in the MC, i.e., the dimension of matrices D_0 and D_1. The mean arrival rate is given by

$$\lambda = \boldsymbol{d} D_1 \mathbf{1}, \tag{2.6}$$

where $\boldsymbol{d}$ is the stationary distribution of the underlying MC, i.e. $\boldsymbol{d}D = 0$ and $\boldsymbol{d}\mathbf{1} = 1$.

2.4 Phase-type distributions

Phase-type (PH) distributions have gained popularity thanks to their ability to capture very general behaviors, while maintaining some of the analytical tractability of the exponential distribution. A PH random variable X [LR99] represents the absorption time in a MC with $n+1$ states, where the states $\{1, 2, \ldots, n\}$ are transient and state 0 is absorbing. We denote it as PH$(\boldsymbol{\tau}, T)$, where $\boldsymbol{\tau}$ is the $1 \times n$ vector of the initial probability distribution, T is the $n \times n$ sub-generator matrix, and the vector $\boldsymbol{t} = -T\mathbf{1}$ holds the absorption rates. The generator matrix Q for the MC in the continuous case for a given representation PH$(\boldsymbol{\tau}, T)$ is given by

$$Q = \begin{bmatrix} T & \boldsymbol{t} \\ \mathbf{0} & 0 \end{bmatrix}.$$

Its cumulative distribution function (CDF) is $F(x) = 1 - \boldsymbol{\tau} \exp(Tx)\mathbf{1}$, for $x \geq 0$, and its expected value is

$$E[X] = -\boldsymbol{\tau} T^{-1} \mathbf{1}. \tag{2.7}$$

This thesis makes extensive use of PH distributions, which serve three main purposes:

- In Chapters 3 and 4, we will show that while the request processing times are exponentially distributed, the *job* processing times follow a PH distribution, whose parameters depend on the overall system state.

- In Chapters 5 and 6, they are used to model request processing times more general than the standard exponential distribution. In addition, the models we introduce obtain PH representations of the service-, waiting- and response-time distributions.

- In Section 5.2, a subset of PH distributions is generalized to model correlated processing times.

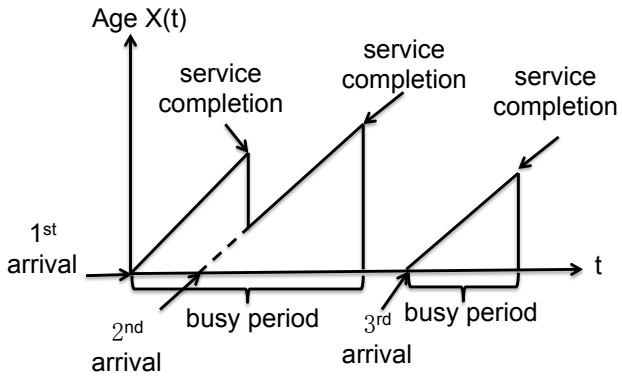


Figure 2.1: Example to illustrate the age process.

Algorithm 1 Computing the response-time distribution for MAP/PH/1 queue

Stage 1: Find service-time distribution PH($\boldsymbol{s}_{\text{ser}}, S_{\text{ser}}$)
Stage 2: Find waiting-time distribution PH($\boldsymbol{s}_{\text{wait}}, S_{\text{wait}}$)
a. Compute T: find S and $A^{(\text{jump})}$, and solve Eq. (2.11)
b. Compute $\boldsymbol{\pi}(0)$: solve Eq. (2.14)
c. Use Eq. (2.18) to find PH($\boldsymbol{s}_{\text{wait}}, S_{\text{wait}}$)
Stage 3: Combine waiting and service times as in Eq. (2.21) to obtain the response-time distribution $PH(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$

2.5 Calculation of the steady state job waiting-time distribution in MAP/PH/1 queues

In this thesis, we are interested in the job response-time distribution, instead of just the mean. One can view the job response time as divided into two parts: (i) a waiting time from the job arrival until it starts processing, and (ii) a service time from the service start until it completes processing. Thus to obtain the *job* response-time distribution, the analysis is divided into two stages: we first obtain the job waiting-time distribution, and then find the job service-time distribution. In the following, we summarize the key steps of existing results in computing the job waiting-time distribution in MAP/PH/1 queues, based on which we generalize to incorporate replication, considering multi-server systems. To help readers navigate the approach, we summarize the key steps in Algorithm 1.

In the system where there is only one server, if a job arrives and finds the server idle, it can start service immediately without waiting. In contrast, if the server is occupied, the waiting time of the current job is the time period between its arrival, and the time its service starts. To

determine the waiting-time distribution, Sengupta [Sen89] proposes to observe the queue only during the **busy periods**, and defining a bivariate Markov process $\{(X(t), J(t))|t \geq 0\}$.

Here the **age** $X(t)$ is the total time-in-system of the job in service, which takes values in $[0, \infty)$, increasing linearly with rate 1 if no service completions occur. The age process $X(t)$ suffers a downward jump when a service completion triggers the start of a new job service, and the value of $X(t)$ after the jump is equal to the waiting time of the job starting service, as it is the only job in service and its age (time-in-system) is equal to its waiting time. Figure 2.1 gives an example of how the age process involves. The first job that arrives sees an empty server, thus it starts service directly, initiating a busy period. From then $X(t)$ increases linearly until this job completes service. Assuming a second job arrives during this busy period, it waits in front of the queue as the server is busy. When the first job completes, the second job starts service, and its total time-in-system is its waiting time, thus $X(t)$ suffers a downward jump, and its new value is equal to the waiting time of the this job. When the second job completes service, as there is no job waiting in the queue, $X(t)$ jumps down to 0, initiating a not-busy period. Similarly, when the third job arrives, it initiates a busy period. The age $X(t)$ thus increases linearly, and jumps down to 0 when this job completes service, assuming no more arrivals during this period.

The *phase* $J(t) = (A(t), S(t))$ holds the joint state of the arrival process $A(t)$ and the service process $S(t)$. Assuming $S(t)$ takes value from the set of server phases N_S with cardinality m_s, then the phase $J(t)$ takes values in a set of size $m = m_a m_s$, where m_a is the number of arrival phases. In this thesis, we follow the $\{(X(t), J(t)\}$ process definition to determine the response-time distribution of jobs with replication. Notice that, while Sengupta [Sen89] considers GI arrivals, we generalize the analysis here to queues with MAP arrivals.

The determination of the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution relies on the stationary distribution $\boldsymbol{\pi}(x)$ of the $(X(t), J(t))$ process, which has a matrix-exponential representation [Sen89]

$$\boldsymbol{\pi}(x) = \boldsymbol{\pi}(0) \exp(Tx), \text{ for } x > 0, \tag{2.8}$$

if process $\{(X(t), J(t))\}$ is positive recurrent. The $m \times m$ matrix T can be found as the solution

to the matrix integral equation

$$T = S \otimes I_{m_a} + \int_0^\infty \exp(Tt)(A^{(\text{jump})} \otimes \exp(D_0 t)D_1)dt, \tag{2.9}$$

where $\otimes$ stands for the Kronecker product, and S and $A^{(\text{jump})}$ are $m_s \times m_s$ matrices that describe the evolution of the service phase process. The off-diagonal entries of S hold the transition rates, between the job service phases N_S, not associated with the start of a new job service, while $A^{(\text{jump})}$ holds the transition rates associated with a new job starting service, during a busy period. The diagonal entries of S are set such that $S + A^{(\text{jump})}$ is the generator of the service phase process [Sen89].

2.5.1 The matrix T

The standard method [Sen89] to solve Eq. (2.9) relies on the numerical integration and is iterative. However, He [He12] proposes to define the matrix

$$L = \int_0^\infty \exp(Tt)(I_{m_s} \otimes \exp(D_0 t))dt \tag{2.10}$$

such that Eq. (2.9) can be written as

$$T = S \otimes I_{m_a} + L(A^{(\text{jump})} \otimes D_1). \tag{2.11}$$

Integrating L by parts, we obtain the following equation for L,

$$TL + L(I_{m_s} \otimes D_0) = -I_m. \tag{2.12}$$

Thus, we compute the T matrix iteratively starting with $T_0 = S \otimes I_{m_a}$, and in the k^{th} iteration solving

$$T_k L_k + L_k(I_{m_s} \otimes D_0) = -I_m \tag{2.13}$$

to find L_k. The iteration continues with $T_{k+1} = S \otimes I_{m_a} + L_k(A^{(\text{jump})} \otimes D_1)$ until convergence is reached. Eq. (2.13) is a Sylvester matrix equation that can be solved in $O(m^3)$ time with the Hessenberg-Schur algorithm [GNVL79].

To complete the matrix-exponential representation $\boldsymbol{\pi}(x) = \boldsymbol{\pi}(0)\exp(Tx)$, we need to find the vector $\boldsymbol{\pi}(0)$, which is the stationary distribution of the phase at the beginning of a busy period. This vector can be found as the solution to the equation

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0) \int_0^\infty \exp(Tt)((S_{\text{ser}}\mathbf{1}) \otimes \exp(D_0 t))dt(-D_0^{-1})(\boldsymbol{s}_{\text{ser}} \otimes D_1), \tag{2.14}$$

where $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ is the PH representation of the job service-time distribution, i.e., the processing-time distribution of the server, which is known beforehand. This equation describes how the phase starts a busy period according to $\boldsymbol{\pi}(0)$, evolves with T and at some time t a service completion occurs with rate $S_{\text{ser}}\mathbf{1}$, which terminates the busy period. Next, the process simply keeps track of the arrival phase, which evolves according to D_0, until a new job arrives with rates in D_1 and the new job starts service according to $\boldsymbol{s}_{\text{ser}}$. After finding L, which is described before in Eq. (2.10), we can re-write Eq. (2.14) as the linear system

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0)L((S_{\text{ser}}\mathbf{1}) \otimes I_{m_a})(-D_0^{-1})(\boldsymbol{s}_{\text{ser}} \otimes D_1), \tag{2.15}$$

which we can readily solve to find $\boldsymbol{\pi}(0)$.

Relying on $(\boldsymbol{\pi}(0), T)$ we can obtain the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution as follows. Let $\boldsymbol{\alpha}_{\text{busy}}$ be the steady-state marginal distribution of the phase process $J(t)$, thus [Sen89]

$$\boldsymbol{\alpha}_{\text{busy}} = \int_0^\infty \boldsymbol{\pi}(t)dt = -\boldsymbol{\pi}(0)T^{-1}. \tag{2.16}$$

Also, let $\boldsymbol{\varphi}$ be the probability that when the current state is $(t + dt, i)$, the next return to a state with age in the interval $[t, t + dt]$ is by way of a downward jump, thus [AM01]

$$\boldsymbol{\varphi} = \int_0^\infty \exp(Tt)(A^{(\text{jump})} \otimes \exp(D_0 t)D_1)dt\mathbf{1} = (T - S \otimes I_{m_a})\mathbf{1}, \tag{2.17}$$

following from Eq. (2.9). Therefore, the PH representation of the waiting time is given by [AM01]

$$\boldsymbol{s}_{\text{wait}} = \gamma \boldsymbol{\alpha}_{\text{busy}} \circ \boldsymbol{\varphi} / ((\boldsymbol{\alpha}_{\text{busy}} \circ \boldsymbol{\varphi})\mathbf{1}), \qquad S_{\text{wait}} = \Lambda^{-1} T' \Lambda, \tag{2.18}$$

where $\circ$ stands for the Hadamard product [Hor90], $'$ denotes the matrix transpose, Λ is a diagonal matrix such that $\Lambda\mathbf{1} = \boldsymbol{s}'_{\text{wait}}$, and γ is the probability that a job has to wait, i.e., the probability that a job finds the server busy. To determine this probability, let η_0 be the number of service completions in a busy period, whose expected value can be obtained as [AM01, Section 7.2]

$$E[\eta_0] = -\frac{\boldsymbol{\pi}(0) T^{-1} (A^{(\text{jump})} \otimes I_{m_a})\mathbf{1}}{\boldsymbol{\pi}(0)\mathbf{1}}. \tag{2.19}$$

Let η_1 be the number of arrivals in a not-busy period, with expected value $E[\eta_1] = 1$, since an arrival during the not-busy period occupies the server, initiating the busy period. Thus the probability that a job has to wait is

$$\gamma = (E[\eta_0] - 1)/(E[\eta_0] - 1 + E[\eta_1]) = 1 - 1/E[\eta_0], \tag{2.20}$$

since, in a cycle made of one busy and one not-busy period, $E[\eta_0] - 1$ is the expected number of jobs that have to wait, and $E[\eta_0] - 1 + E[\eta_1]$ is the expected number of job arrivals.

After the PH representation $\text{PH}(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution has been obtained, the PH representation $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$ of the response-time distribution is

$$\boldsymbol{s}_{\text{res}} = [\boldsymbol{s}_{\text{wait}} \quad (1 - \boldsymbol{s}_{\text{wait}}\mathbf{1})\boldsymbol{s}_{\text{ser}}], \quad S_{\text{res}} = \begin{bmatrix} S_{\text{wait}} & (-S_{\text{wait}}\mathbf{1})\boldsymbol{s}_{\text{ser}} \\ \mathbf{0} & S_{\text{ser}} \end{bmatrix}, \tag{2.21}$$

where $\text{PH}(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ is the given PH representation of the server's processing-time distribution. Note that Eq. (2.21) applies as the waiting and service processes are independent. This representation captures that for some jobs the response time is made of a first period of waiting followed by a period of service, while others start service directly. With this representation we can readily compute response-time percentiles and moments.

Based on the above existing results, we introduce in this thesis novel stochastic models to eval-

uate how speculative replication affects job response-time distribution in multi-server systems, which allow us to characterize the conditions under which replication effectively reduces the response times.

2.6 Related Work

Request replication has been considered in [DB13, SLR16, VGM+13, AGSS12, Sny10, GZD+15, JSW15, RAWY15, GHBSW16] as an efficient way to combat the variability in latency. Trace-driven simulations in [AGSS12] show that processing replicas of requests simultaneously is effective in mitigating the high average response time of long-running outliers. Joshi et. al. [JSW15] analyze the impact of replication on the mean response time and the cost of computing resources, and show that this impact depends on the processing-time distribution. Vulimiri et. al. [VGM+13] propose a queueing model to derive the mean response time as a function of system utilization and service-time distribution, and approximate the threshold load under which replication improves mean latency.

Most existing works [DB13, SLR16, GZD+15, JSW15] consider the adoption of canceling to limit the additional load introduced by the replicas. In fact, it is common to assume that redundant replicas can be canceled at no cost. Dean and Barroso [DB13] observe that adopting replication with canceling in a benchmark experiment reduces the latency substantially, while introducing only a slight additional load. In fact, canceling plays a key role in obtaining the scarce analytical solutions available for request replication [GZD+15, JSW15]. For instance, canceling enables [JSW15] to reduce the problem to a single M/G/1 queue in the early-canceling case or to an approximate M/G/C queue in the late-canceling one. Similarly, canceling is key in obtaining the solution to the Markov chain model in [GZD+15]. From an analytical standpoint, the cases without canceling considered in this thesis are more challenging than their counterpart with canceling, where replicas of a request always finish service at the same time, adding a synchronization point that simplifies the system dynamics. The absence of canceling allows servers to evolve asynchronously, requiring a more complex analysis. This is also reflected in

the few existing results that consider the cost of canceling, as these focus on fairly limited scenarios, e.g., 2-server systems with a centralized queue, Poisson arrivals, and exponential processing times [LPR15]. In contrast, in this thesis we consider in Section 5.3 where canceling operation may require a significant amount of time, compared to the replica processing time, and in Chapter 5 cases where canceling is hard or even infeasible to implement.

In many applications, requests are fairly homogeneous and their processing times depend largely on the server, leading to a degree of independence between replicas' service times. However, in other applications, processing time depends on the intrinsic details of the request, which impacts all of its replicas. In such cases correlation among the replicas of a request becomes significant; whilst commonly ignored [VGM+13, GZD+15], it has only recently been taken into account in [JSW15, GHBSW16]. In contrast, Section 5.2 considers cases of both independent and correlated processing times among replicas of the same request.

We should also highlight that most existing works focus on either modeling or system experimentation. The only exception is [VGM+13], which studies the threshold load by means of approximate models, and experiments with different applications to confirm that this threshold load for one additional replica is between 25% and 50%. To the best of our knowledge, the model introduced in Section 5.2 is the first time a model for request replication is able to provide response time mean and percentiles, and its predictions are validated against system measurements.

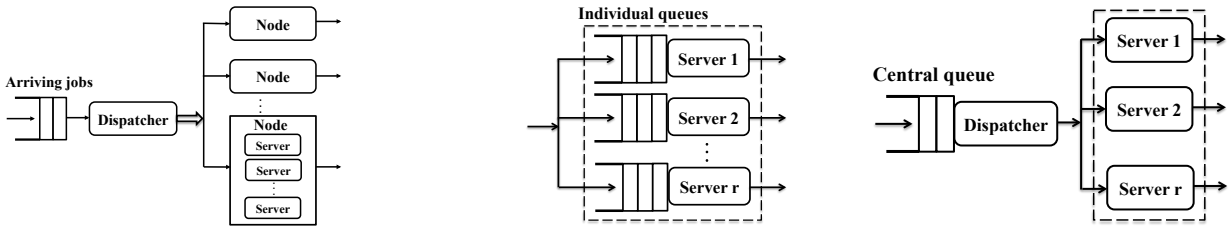
Chapter 3

Fault-tolerance and Latency-reduction via Replication: the Distributed Set-up

Although much effort has been spent on optimizing the performance of computing clusters [GHBDL09, GJW11, Mit13], requests unavoidably experience failures or delays, degrading the offered quality of service. In this chapter, we evaluate the effectiveness of the speculative replication strategy has on the system reliability and request latency in systems where requests are subject to failures during service. In particular, to limit the additional load introduced by replication, we adopt the so called *replica cancellation* scheme: after retrieving the result from the first replica that completes, all other outstanding replicas are cancelled immediately. Using this approach, the resources used by a job and its replica are freed as soon as one of them completes service.

3.1 Reference model

We consider a server farm consisting of a number of distributed, homogeneous, and independent servers, which process incoming requests with exponential service rate μ. The request in service is subject to failure, where the time-to-failure is exponentially distributed with rate α, a common assumption in reliability engineering [ZL09, WSJ07]. In the case of a request failure, we assume



(a) Server farm (b) The distributed set-up (c) The centralized set-up

Figure 3.1: Reference models. A server farm is split in nodes, as in (a). The set-up of each node can be as in (b) or in (c).

that the server is not affected by the failure, and continues serving the next request in front of the queue.

To improve the reliability, $r-1$ extra replicas are adopted for each arriving request, i.e., a total of r replicas of a request are submitted to the system. In particular, $r = 1$ represents the case where no replication is adopted. We denote by *request/replica* the original request or any of its replicas, and by *job* the set of replicas for a single request, where the number of replicas r is also referred to as the *replication level*. To reduce unnecessary workload, the system replies with the result from whichever replica completes successfully first, and immediately cancels all the other outstanding replicas in the same job. The canceling overhead, which is the time to remove all the replicas in a job, is assumed to be negligible.

To take advantage of replication, we consider a server farm consisting of n distributed computing nodes, and a central scheduler that assigns jobs to the processing nodes in a round-robin or random fashion, as shown in Figure 3.1(a). The node can have distributed queues in front of each of the servers, as shown in Figure 3.1(b). An alternative set-up is to have a centralized queue in front of all the servers, as shown in Figure 3.1(c), where all the replicas of the incoming requests form a single queue in the order of arrival and join the next server that becomes available with first-come first-served (FCFS) scheduling. For this distributed case, each node is composed of r independent and statistically identical processing servers, thus serving a job of replication level r, by processing each of the r replicas in each server. Each of the r replicas thus joins the end of the queue at one of the r servers, and is processed with FCFS scheduling. For both models, when one of the replicas in a job completes service successfully, it immediately cancels all its outstanding siblings, *either waiting or in service*. However, if a replica fails during

service, it leaves the system without influencing its siblings.

In this chapter we develop stochastic models to determine the response-time distribution for systems with the distributed set-up. In Section 3.2 we consider the set-up with two replicas, where we assume a Markovian system and use the generating function method to determine the Laplace transform of the response-time probability distribution, jointly with the probability that not both replicas fail. We then generalize the number of replicas, and develop in Section 3.3 an approximate stochastic model to determine the request response-time distribution. Validation against simulation indicates that this stochastic model is accurate for not only the mean response time but also its percentiles, which are particularly relevant for deadline-driven applications. Further, we show that in the distributed set-up, replication with canceling has the potential to reduce response times, even at high utilization levels. These finding supports the use of replication as an effective mechanism for both fault-tolerance and delay-reduction.

Notice that in this thesis we focus on a single computing node that implements replication, as the analysis can be easily extended to the multi-node case to evaluate the overall performance of the server farm, as described below.

3.1.1 The multi-node system

In this thesis, we assume that jobs that arrive at the dispatcher are assigned to the computing nodes in either round-robin or random fashion. The advantage of round-robin scheduling is that jobs are distributed evenly among the nodes, and no communication is required between the nodes and the dispatcher about the nodes state. Under either round-robin or random allocation, if the overall arrival process is a MAP(D_0,D_1), the arrival process to each node is also a MAP. Under round-robin allocation, the parameters are

$$C_0 = \begin{bmatrix} D_0 & D_1 & . & \cdots \\ . & D_0 & D_1 & \cdots \\ \vdots & \vdots & \ddots & \ddots \\ . & . & \cdots & D_0 \end{bmatrix}, \quad C_1 = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ D_1 & 0 & \cdots & 0 \end{bmatrix}. \tag{3.1}$$

These matrices are of size nm_a, where n is the number of computing nodes in this group. Under the random allocation, the arrival process has parameters

$$C_0 = D_0 + (1-p)D_1, \quad C_1 = pD_1,$$

where $p = 1/n$. As the arrival process to each computing node is still a MAP, it fits into the proposed models.

3.2 The case with $r = 2$ replicas

In this section, we focus on systems with the distributed set-up, where we consider a single computing node that implements replication with canceling and operates with $r = 2$ individual queues, i.e., one extra replica is adopted for each request. We consider a pair of FCFS queues, where a single Poisson arrival process of rate λ sends a request to both queues at an arrival instant. When a request completes service, it cancels its outstanding partners, either waiting in the queue or in service. However, it may be that its partner had already failed, in which case no action is taken on the service completion.

3.2.1 Conditional request response-time distribution

In conventional queuing terminology, we seek the joint probability that the two requests in a tagged pair do not both fail, i.e., one completes service in one of the queues, and their response time R does not exceed a positive real value t, conditioned on the numbers of requests ahead of each of the tagged pair, N_1 and N_2, in each queue initially – e.g. at the tagged pair's arrival instant. If the tagged replicas both fail, $R > t$ for all $t > 0$, so that $R = \infty$ with probability one. For integers $m, n \geq 0$, we define

$$F_{mn}(t) = \Pr(R \leq t \mid N_1 = m, N_2 = n),$$

and obtain a recurrence formula for $F_{mn}(t)$, which we solve via a generating function of its Laplace transform. We then decondition against the joint stationary distribution of N_1 and N_2 at the arrival instant of the tagged pair. Let $f_{mn}(t)$ be the derivative of $F_{mn}(t)$ with respect to t, and

$$L_{mn}(s) = (\mathcal{L}F_{mn})(s)$$

be the Laplace transform of $F_{mn}(t)$. Noting that, $F_{mn}(t) = F_{nm}(t)$ and $L_{mn}(s) = L_{nm}(s)$ for all $m, n \geq 0$ by the symmetry of the two queues.

Instead of keeping track of the lengths of both queues, we focus on the difference between the queue-lengths, i.e., we focus on the length of the shorter queue, $n \geq 0$, and the difference $k = m - n \geq 0$, assuming $m \geq n$. For example, difference $k = 4$ means the queue lengths are $(4 + n, n)$ or $(n, 4 + n)$. Thus we have $F_{mn} = F_{n+k,n}$ and $L_{mn} = L_{n+k,n}$. Let

$$G(x, y) = \sum_{n=0}^{\infty} \sum_{k=0}^{\infty} L_{n+k,n} x^n y^k = \sum_{k=0}^{\infty} G_k(x) y^k,$$

$\forall x, y, s$ with $|x| < 1, |y| < 1$ and real values $s \geq 0$, where n is always the queue length of the shorter queue, and

$$G_k(x) = \sum_{n=0}^{\infty} L_{n+k,n} x^n$$

is the coefficient of y^k in $G(x, y)$. When one of the tagged pair of replicas has failed, the problem reduces to a single queue. Focusing on the only queue with the tagged replica, and letting $k = m$ for this case, we define $F_{m,-1}(t) = F_{k,-1}(t) = F_{-1,k}(t) = \Pr(R' \leq t \mid N_1 = k)$ for $k \geq 0$, where -1 indicates the failure of the tagged replica in the queue. R' is the response time random variable of queue 1 considered in isolation when there are m requests ahead of a single tagged replica on its arrival. Similarly to the previous notation, $f_{k,-1}(t) = F'_{k,-1}(t)$ and $L_{k,-1} = \mathcal{L}F_{k,-1}$. We then define the corresponding one-parameter generating functions:

$$G(0, y) = \sum_{k=0}^{\infty} L_{k,0} y^k, \quad G_{-1}(y) = \sum_{k=0}^{\infty} L_{k,-1} y^k.$$

Assuming that the tagged pair arrives at time 0, let $N_1(t), N_2(t)$ be the numbers of requests

ahead of the tagged replicas in queue 1 and queue 2 respectively at time t. Then we have:

$$\begin{aligned} F_{mn}(t+h) &= \Pr(R \le t+h \mid N_1(0)=m, N_2(0)=n) \\ &= \sum_{i,j=0}^{\infty} \Pr(R \le t+h \mid N_1(0)=m, N_2(0)=n, N_1(h)=i, N_2(h)=j) \\ &\quad \times \Pr(N_1(h)=i, N_2(h)=j \mid N_1(0)=m, N_2(0)=n) \end{aligned}$$

by the law of total probability. The terms $\Pr(N_1(h)=i, N_2(h)=j \mid N_1(0)=m, N_2(0)=n)$ correspond to the possible events that can occur during the interval $(0,h]$ for $h>0$, which we take to be infinitesimal.

Proposition 1. *For* $|x|<1, |y|<1$,

$$G(x,y) = \frac{(2\mu+2\alpha+s-(\mu+\alpha)y)G(0,y)+(s+2\mu+2\alpha-\mu x-\alpha x/y)G_0(x)-\alpha x G_1(x)-(s+2\mu+2\alpha)L_{0,0}}{2\mu+2\alpha+s-\mu x-(\alpha+\mu)y-\alpha x y^{-1}} \tag{3.2}$$

Proof. When $m>n>0$, e.g. $m=n+k$ with $k\ge 1, n\ge 1$, the possible events in the interval $(0,h]$ have the following probabilities, by the memoryless property of the exponential random variable:

- a departure of the replica in service in queue 1, with probability $\mu h+o(h)$ if $i=m-1, j=n$;
- a departure of the replica in service in queue 2, with probability $\mu h+o(h)$ if $i=m-1, j=n-1$, since the departure of the replica in the shorter queue will cancel its partner in the longer queue;
- a failure of the replica in service in queue 1, with probability $\alpha h+o(h)$ if $i=m-1, j=n$;
- a failure of the replica in service in queue 2, with probability $\alpha h+o(h)$ if $i=m, j=n-1$;
- no change to the states of the queues with probability $1-2\alpha h-2\mu h+o(h)$ if $i=m, j=n$.

All other values of i and j give a second-order contribution as more than one event in the interval $(0, h]$ would be required. Thus, $F_{mn}(t+h)$ satisfies:

$$\begin{aligned}
F_{mn}(t+h) &= \mu h \Pr(R \le t+h \mid N_1(0)=m, N_2(0)=n, N_1(h)=m-1, N_2(h)=n) \\
&\quad + \mu h \Pr(R \le t+h \mid N_1(0)=m, N_2(0)=n, N_1(h)=m-1, N_2(h)=n-1) \\
&\quad + \alpha h \Pr(R \le t+h \mid N_1(0)=m, N_2(0)=n, N_1(h)=m-1, N_2(h)=n) \\
&\quad + \alpha h \Pr(R \le t+h \mid N_1(0)=m, N_2(0)=n, N_1(h)=m, N_2(h)=n-1) \\
&\quad + (1-2\mu h - 2\alpha h) \Pr(R \le t+h \mid N_1(0)=N_1(h)=m, N_2(0)=N_2(h)=n) \\
&\quad + o(h)
\end{aligned}$$

By the Markov property applied at time h and the residual life property of exponential random variables, the remaining response time of the tagged pair after time h, $R_h = R - h$ say, is distributed as the full response time of a hypothetical arrival-pair at time h conditioned on there being $N_1(h)$ and $N_2(h)$ requests ahead in queues 1 and 2 respectively, c.f. [HP93]. The last conditional probability (for instance) then simplifies to

$$\Pr(R_h \le t \mid N_1(h)=m, N_2(h)=n) = \Pr(R \le t \mid N_1(0)=m, N_2(0)=n) = F_{mn}(t)$$

Thus we have

$$\begin{aligned}
F_{mn}(t+h) &= \mu h F_{m-1,n}(t) + \mu h F_{m-1,n-1}(t) + \alpha h F_{m-1,n}(t) + \alpha h F_{m,n-1}(t) \\
&\quad + (1-2\mu h - 2\alpha h) F_{mn}(t) + o(h)
\end{aligned}$$

Rearranging, dividing by h, taking the limit $h \to 0$ and omitting the argument t for brevity, we obtain

$$f_{mn} = \mu F_{m-1,n} + \mu F_{m-1,n-1} + \alpha F_{m-1,n} + \alpha F_{m,n-1} - (2\mu + 2\alpha) F_{mn}$$

Taking the Laplace transform, noting that $(\mathcal{L} f_{mn})(s) = s L_{mn}(s)$ and writing $n+k$ for m, we obtain

$$(2\mu + 2\alpha + s) L_{n+k,n} = (\alpha + \mu) L_{n+k-1,n} + \alpha L_{n+k,n-1} + \mu L_{n+k-1,n-1}$$

Multiplying by x^n and summing from $n = 1$ to ∞, we find

$$(2\mu+2\alpha+s-\mu x)G_k(x) = (\alpha+\mu)G_{k-1}(x)+\alpha x G_{k+1}(x)+(2\mu+2\alpha+s)L_{k,0}-(\mu+\alpha)L_{k-1,0} \quad (3.3)$$

Multiplying by y^k, summing from $k = 1$ to ∞ and rearranging now gives Eq. (3.2). □

Lemma 3.2.1.

$$L_{0,-1} = \frac{\mu}{s(s+\mu+\alpha)} \quad (3.4)$$

Proof. Proceeding as in the proof of Proposition 1, for $m = 0, n = -1$, we have

$$F_{0,-1}(t+h) = \mu h \cdot 1 + \alpha h F_{-1,-1}(t) + (1-\mu h-\alpha h)F_{0,-1}(t) + o(h)$$

The term $\mu h \cdot 1$ corresponds to the tagged replica completing service in queue 1. The term $F_{-1,-1}(t) = 0$ corresponds to the failure of the tagged replica in both queues, with zero probability of completing service in less than time t for any $t > 0$. Then we obtain $L_{0,-1}$, proceeding analogously to the proof of Proposition 1. □

Lemma 3.2.2.

$$L_{0,0} = \frac{2\mu s + 2\mu^2 + 4\alpha\mu}{s(s+\mu+\alpha)(s+2\mu+2\alpha)} \quad (3.5)$$

Proof. For $m = 0, n = 0$, we have, similarly to above,

$$(2\mu+2\alpha+s)L_{0,0} = \alpha L_{-1,0} + \alpha L_{0,-1} + 2\mu/s = 2\alpha L_{0,-1} + 2\mu/s \quad (3.6)$$

The result follows by substituting Eq. (3.4) into Eq. (3.6). □

Lemma 3.2.3. *For* $|y| < 1$,

$$G_{-1}(y) = \frac{\mu}{s((\mu+\alpha+s)-y(\mu+\alpha))} \quad (3.7)$$

Proof. For $m > 0, n = -1$, we have

$$(\mu+\alpha+s)L_{k,-1} = (\alpha+\mu)L_{k-1,-1} \quad (3.8)$$

Multiplying by y^k, summing from $k = 1$ to ∞, we find

$$(\mu + \alpha + s)(G_{-1}(y) + L_{0,-1}) = (\alpha + \mu)yG_{-1}(y).$$

Combining with Eq. (3.4), and rearranging gives Eq. (3.7). □

Lemma 3.2.4. *For $|y| < 1$,*

$$G(0, y) = \frac{(s + 2\alpha + 2\mu)L_{0,0} + \alpha G_{-1}(y) - \alpha L_{0,-1} + \mu y(s(1 - y))^{-1}}{s + 2\mu + 2\alpha - (\mu + \alpha)y} \tag{3.9}$$

Proof. For $m > 0, n = 0$, we have, following the previous proof method:

$$F_{k,0}(t + h) = \mu h F_{k-1,0}(t) + \mu h \cdot 1 + \alpha h F_{k-1,0}(t) + \alpha h F_{k,-1}(t) + (1 - 2\mu h - 2\alpha h)F_{k,0}(t) + o(h)$$

Similar steps to before yield,

$$(2\mu + 2\alpha + s)L_{k,0} = (\alpha + \mu)L_{k-1,0} + \alpha L_{k,-1} + \mu/s \tag{3.10}$$

and the result follows from Lemmas 3.2.1, 3.2.2 and 3.2.3. □

Proposition 2. *For $|x| < 1$,*

$$G_0(x) = \frac{(s/2 + \mu + \alpha)L_{0,0} - (2\mu + 2\alpha + s - (\alpha + \mu)y_0(x))G(0, y_0(x))}{s/2 + \alpha + \mu - \alpha x/y_0(x)}, \qquad \forall x \in (0, 1). \tag{3.11}$$

where $y_0(x) = (s + 2\alpha + 2\mu - \mu x - \sqrt{(s + 2\alpha + 2\mu - \mu x)^2 - 4\alpha x(\alpha + \mu)})/(2(\alpha + \mu))$.

Proof. For $l = m > 0$, i.e., $k = 0, n \geq 1$, we have:

$$(2\mu + 2\alpha + s)L_{n,n} = 2\alpha L_{n,n-1} + 2\mu L_{n-1,n-1}$$

Multiplying by x^n, summing from $n = 1$ to ∞ then gives

$$(2\mu + 2\alpha + s - 2\mu x)G_0(x) = 2\alpha x G_1(x) + (2\mu + 2\alpha + s)L_{0,0} \tag{3.12}$$

Now consider the expression for $G(x, y)$ in Eq. (3.2). Since it is analytic in the unit disks of both x and y, the numerator must vanish wherever the denominator is equal to 0. The zeros of the denominator are those of the quadratic

$$\text{Den} \equiv (\alpha + \mu)y^2 - (s + 2\mu + 2\alpha - \mu x)y + \alpha x = 0$$

When $y = 0$, Den $= \alpha x > 0$ for nonzero failure rate α, and when $y = 1$, Den $= -(\alpha + \mu)(1 - x) - s < 0$ for $0 < x < 1$. When $y \to +\infty$, Den > 0 since the equation is dominated by y^2. Thus this equation has one (smaller) root in interval $(0, 1)$ and another in $(1, \infty)$. Given some $x \in (0, 1)$, let the smaller root for y be $y_0(x) \in (0, 1)$, namely:

$$y_0(x) = \frac{s + 2\alpha + 2\mu - \mu x - \sqrt{(s + 2\alpha + 2\mu - \mu x)^2 - 4\alpha x(\alpha + \mu)}}{2(\alpha + \mu)}.$$

To make $G(x, y)$ analytic, the pair $(x, y_0(x))$ must also be a root of the numerator of Eq. (3.2), i.e. we must have

$$(2\mu + 2\alpha + s - (\mu + \alpha)y_0(x))G(0, y_0(x)) + (s + 2\mu + 2\alpha - \mu x - \alpha x/y_0(x))G_0(x) - \alpha x G_1(x) - (s + 2\mu + 2\alpha)L_{0,0} = 0.$$

Substituting $G_1(x)$ using $G_0(x)$ following Eq. (3.12), we obtain Eq. (3.11). □

3.2.2 Formulation based on difference between queue-lengths

Under the FCFS discipline, the unconditional probability distribution of response time at equilibrium, $R(t)$, is $R(t) = P(R \leq t) = \sum_{n\geq 0}\sum_{m\geq 0} \pi_{mn}F_{mn}(t) = \sum_{n\geq 0}\sum_{k\geq 0} \pi_{n+k,n}F_{n+k,n}(t)$ (for $m \geq n, k \geq 0$), which has Laplace transform

$$R^*(s) = \sum_{n\geq 0}\sum_{k\geq 0} \pi_{n+k,n}L_{n+k,n}(s),$$

where $\pi_{n+k,n}$ is the steady state probability that the joint state of the queues is $(n + k, n)$, i.e., when the length of the shorter and the longer queue is n and $n+k$, respectively. We then essen-

Table 3.1: Mean and Standard deviation of response times with $\mu = 1$: simulation, and SEM.

α	λ	**Mean**		**Standard deviation**	
		Simulation[95%]	SEM/%Err	Simulation[95%]	SEM/%Err
0.001	0.1	0.5270 ±0.0003	0.5268/0.00	0.5279 ±0.0002	0.5277/0.04
	0.5	0.6669 ±0.0004	0.6671/0.00	0.6674 ±0.0003	0.6677/0.04
	0.9	0.9091 ±0.0004	0.9095/0.00	0.9101 ±0.0004	0.9098/0.03
0.01	0.1	0.5310 ±0.0003	0.5310/0.04	0.5394 ±0.0002	0.5395/0.00
	0.5	0.6707 ±0.0005	0.6708/0.00	0.6762 ±0.0003	0.6763/0.00
	0.9	0.9121 ±0.0005	0.9127/0.07	0.9151 ±0.0004	0.9155/0.04
0.1	0.1	0.5558 ±0.0004	0.5555/0.05	0.6042 ±0.0002	0.6037/0.08
	0.5	0.6906 ±0.0005	0.6901/0.07	0.7243 ±0.0004	0.7234/0.00
	0.9	0.9236 ±0.0006	0.9223/0.03	0.9409 ±0.0004	0.9392/0.18
	0.99	0.9987 ±0.0007	0.9997/0.10	1.0115 ±0.0004	1.0128/0.13
0.5	0.1	0.5204 ±0.0004	0.5203/0.00	0.5711 ±0.0004	0.5711/0.00
	0.5	0.6242 ±0.0005	0.6236/0.10	0.6647 ±0.0003	0.6636/0.17
	0.9	0.7852 ±0.0005	0.7857/0.06	0.8076 ±0.0004	0.8087/0.14
	1.35	1.1347 ±0.0007	1.1359/0.11	1.1260 ±0.0005	1.1288/0.25
1	0.2	0.4470 ±0.0004	0.4469/0.00	0.4769 ±0.0002	0.4773/0.08
	1.0	0.6351 ±0.0005	0.6352/0.00	0.6514 ±0.0003	0.6514/0.00
	1.8	1.1446 ±0.0008	1.1468/0.01	1.1165 ±0.0006	1.1202/0.33

tially use the difference k to modulate the Markov process followed by the shorter queue length using the spectral expansion method. Notice that the probability that a positive customer completes service at either of the servers is the marginal probability $R(\infty) = sR^*(s)|_{s=0}$.

The spectral expansion method (SEM) of [Mit02] is ideally suited to calculating the joint equilibrium probabilities in the difference formulation, in which the system state is described by a pair of random variables, (K, N); K is a control variable that determines the transition rates in the Markov process followed by N. Here, K is the difference and N is the length of the shorter queue. The random variable K must be finite and so we truncate the state space thus K is at maximum a constant C. The limit C introduces an approximation, the goodness of which depends on the system parameters. For instance, when the failure rate α is small compared to the service rate μ, the probability of a large difference K is small, since successful service completions are more likely to occur, activating the canceling mechanism, which keeps the queues more synchronized. With a larger failure rate, we may expect a larger difference. However, due to the canceling mechanism, the probability of large differences between queue lengths stays small, even at high utilization levels. From our experiments, a truncation at maximum difference $C = 5$ was found sufficient.

To evaluate the accuracy of our approximation, we compare its results against those estimated

from simulation experiments. As there is no standard simulation tool that can simulate failures, replication and canceling, we implemented an event-based simulation in Java, where the job inter-arrival times, service and failure times are generated according to the exponential distribution. In Table 3.1, we compare the mean and standard deviation of response time obtained from simulation, with the SEM results ($C = 5$). Here we set the service rate $\mu = 1$, and consider different failure rate α, and arrival rate λ, thus achieving different reliability and utilization levels. For each setting, the simulations were run for 5,000 times with 500,000 samples each time. Here we can see that our model shows a high degree of accuracy for all the cases.

In this section, we analyzed the effectiveness of replication schemes, considering the case of two replicas. The approximation method proposed – focusing on the difference between the queue lengths, achieves high accuracy. However, this solution, which involves using generating functions to solve the request response-time distribution, is limited by its complexity. In the next section, we investigate the case with $r \geq 2$ replicas using the matrix-analytic method.

3.3 The case with $r \geq 2$ replicas

Analyzing the model with $r \geq 2$ and distributed queues is challenging, as the synchronized arrivals of replicas to all queues correlate their dynamics, while individual replicas fail asynchronously. We therefore rely on the matrix-analytic method to develop an approximate stochastic model to determine the job response-time distribution, capturing the dynamics introduced by failures, replication and canceling. Although approximate, validation results indicate that our model is accurate for not only the mean response time but also its percentiles, a significant advantage since mean response time guarantees are not sufficient in many situations, in particular for deadline-driven applications. Further, we show that in the distributed set-up replication with canceling has the potential to reduce the response times, even under relatively high utilization. The effect of replication actually depends on the specific response time percentile evaluated, especially in low-reliability scenarios. This effect is more uniform across the response-time distribution when the request reliability is high or when more than one extra

replica is adopted.

To this end we start by obtaining the waiting-time and service-time distributions separately, both of which have Phase-type (PH) representations. For each job, composed of r replicas, the waiting time is the period between its arrival and the time the first of its replicas starts service, while the service time starts when the waiting period ends and concludes when one of the replicas completes service successfully, or when all replicas fail.

3.3.1 The waiting-time distribution

Different from the single-server scenario considered in Section 2.5, the system considered in this section is more complicated as it is composed of $c \geq 1$ servers. In the system with multiple servers, we follow the definitions in [AM01], where an *all-busy period* refers to the time period when all the servers are busy, which terminates when one of the servers becomes idle and its queue is empty. This marks the start of a *not-all-busy* period, where at least one server is idle and terminates when a job arrives, submitting one replica to each server, initiating an all-busy period. To determine the waiting-time distribution, we observe the queues only during the *all-busy* periods, and define a bi-variate Markov process $\{X(t), J(t)|t \geq 0\}$ [AM01]. As the single-server case, the *phase process* $J(t) = (A(t), S(t))$ still holds the state of the arrival and service processes, where the arrival process is a MAP with m_a phases and parameters (D_0, D_1). The *age process* $X(t)$ is the total time-in-system of the *youngest* job in service, which may increase linearly with rate 1, if no *job* service completions occur. Note that, during the all-busy period, a new job starts service only if the replica in service in the *shortest queue* completes service or fails. This is because for the replicas waiting in other queues, one or more of their siblings have already failed. Thus in case of a service completion or a failure in the shortest queue, a new job starts service and a downward jump in $X(t)$ occurs, as the new job becomes the youngest in service and its age (time-in-system) is equal to its waiting time.

To model the service process, we first order the queue lengths in ascending order, i.e., $(q_1, q_2, \ldots, q_r)$ with $q_i \leq q_j$ for $i < j$ and $1 \leq i, j \leq r$, where the queue length includes jobs waiting and in service. During the all-busy period, the shortest queue length must be positive, i.e., $q_1 > 0$,

Table 3.2: Transition rates for matrix S

From	To	Rate	Range
$(d_1,\ldots,d_i,d_{i+1},\ldots,d_{r-1})$	$(d_1,\ldots,d_i{-}1,d_{i+1},\ldots,d_{r-1})$	μ	$d_i>0,\ d_{i+1}>0, i\geq 1$
$(d_1,\ldots,d_i,d_{i+1},\ldots,d_{r-1})$	$(d_1,\ldots,d_i{-}1,\max\{C,d_{i+1}{+}1\},\ldots,d_{r-1})$	α	$d_i>0,\ d_{i+1}>0, i\geq 1$
$(d_1,\ldots,d_i,0,\ldots,0,d_k,\ldots,d_{r-1})$	$(d_1,\ldots,d_i{-}1,0,\ldots,0,d_k,\ldots,d_{r-1})$	$(k{-}i)\mu$	$d_i>0,\ d_j=0 \quad \forall i<j<k$
$(d_1,\ldots,d_i,0,\ldots,0,d_k,\ldots,d_{r-1})$	$(d_1,\ldots,d_i{-}1,1,0,\ldots,0,d_k,\ldots,d_{r-1})$	$(k{-}i)\alpha$	$d_i>0,\ d_j=0 \quad \forall i<j<k$

Table 3.3: Transition rates for matrix $A^{(\text{jump})}$

From	To	Rate	Range
$(d_1,d_2,\ldots,d_{r-1})$	$(d_1,d_2,\ldots,d_{r-1})$	μ	$d_1>0$
$(d_1,d_2,\ldots,d_{r-1})$	$(\max\{C,d_1{+}1\},d_2,\ldots,d_{r-1})$	α	$d_1>0$
$(0,\ldots,0,d_i,\ldots,d_{r-1})$	$(0,\ldots,0,d_i,\ldots,d_{r-1})$	$i\mu$	$d_i>0,\ d_j=0 \quad \forall j<i$
$(0,0,\ldots,0,d_i,\ldots,d_{r-1})$	$(1,0,\ldots,0,d_i,\ldots,d_{r-1})$	$i\alpha$	$d_i>0,\ d_j=0 \quad \forall j<i$

while during the not-all-busy period the shortest queue must be empty, i.e., $q_1=0$. Further, we focus on the differences between two consecutive queues after ordering, defined as $(d_1,\ldots,d_{r-1})$, where $d_i=q_{i+1}-q_i$ for $1\leq i\leq r{-}1$. The queue-length difference is unbounded in principle, but, to keep the phase space finite, we introduce an upper bound $C<\infty$ as in Section 3.2, such that the difference is at most C. As a result, the service process $S(t)$ takes values in the set $N_N=\{(d_1,\ldots,d_{r-1})|d_i\in\{0,\ldots,C\} \text{ for } 1\leq i\leq r-1\}$, the cardinality of which is $m_s=(C+1)^{r-1}$ for a system with r queues and an upper bound of C. We evaluate later in Section 3.3.3 the accuracy of this approximation and explore the selection of the limit C. The phase process $J(t)$ thus takes $m=m_am_s$ different values, where m_a is the number of arrival phases.

To determine the PH representation $(\boldsymbol{s}_{\text{wait}},S_{\text{wait}})$ of the waiting-time distribution, we rely on the stationary distribution $\boldsymbol{\pi}(x)$ of the $(X(t),J(t))$ process, which involves to solve the matrix T in Eq. (2.9) as defined in Section 2.5. As mentioned above, only the service completion or failure of the replica in service in the shortest queue triggers the start of a new *job* service, thus transitions in the shortest queue correspond to matrix $A^{(\text{jump})}$, while transitions in other queues correspond to matrix S. Table 3.2 shows the transition rates in S. The first row considers the case where the replica in service in the $(i{+}1)^{\text{th}}$ shortest queue *completes service*, canceling its partners in all the queues longer than queue $(i{+}1)^{\text{th}}$, resulting in the difference between the $(i{+}1)^{\text{th}}$ and i^{th} shortest queues decreasing by 1, while other differences remain unaffected. The second row considers the case where the replica in the $(i{+}1)^{\text{th}}$ shortest queue *fails*, decreasing

by 1 the difference between the $(i{+}1)^{\text{th}}$ and i^{th} shortest queues, while the difference between the $(i{+}2)^{\text{th}}$ and $(i{+}1)^{\text{th}}$ shortest queues increases by 1, but bounded by C. In the previous two cases we assumed that $d_{i+1} > 0$, thus ensuring that the $(i{+}1)^{\text{th}}$ shortest queue is actually a single queue. The third and fourth rows consider the case where $d_{i+1} = \cdots = d_{k-1} = 0$, such that from the $(i{+}1)^{\text{th}}$ to the k^{th} shortest queues have the same length. Thus, the transition rates in the third row reflect that a service completion in any of these queues triggers the same transition. The fourth row considers the same condition for the case of replica failures. Table 3.3 considers similar conditions but for the matrix $A^{(\text{jump})}$. The first and second rows consider a service completion or failure of the replica in the shortest queue, thus initiating a new job service. In case of a service completion, the replica that completes service cancels all its partners, thus the differences between queue-lengths remain unchanged. On the other hand, a failure in the shortest queue leads d_1 to increase by 1, but bounded by the limit C. The last two rows consider the case where there are multiple shortest queues.

With matrices S and $A^{(\text{jump})}$, we could find the matrix T by solving Eq. (2.9). And after the matrix T is found, we again need to determine the steady state distribution $\boldsymbol{\pi}(0)$ of the phases at the beginning of an all-busy period. To find $\boldsymbol{\pi}(0)$, we need to connect the not-all-busy and the all-busy periods [AM01]. Compared to the all-busy period, in the not-all-busy period the shortest queue is empty, $q_1 = 0$, but the differences between queue-lengths can be modeled just as in the all-busy period. Thus, during the not-all-busy period we keep track of the arrival and services phases $J_N(t) = (A(t), S_N(t))$, with the service phase $S_N(t) = S(t)$ as the all-busy period, taking values in the set S_S.

Note that a single arrival during the not-all-busy period is enough to initiate an all-busy period, as its replicas are sent to each queue. Thus the vector $\boldsymbol{\pi}(0)$, which is the distribution of the phase $J(t)$ at the beginning of an all-busy period, solves

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0) \int_0^\infty \exp(Tu)(A^{(\text{jump})} \otimes \exp(D_0 u))du(S_{\text{not-all-busy}} \oplus D_0)^{-1}(I_{m_s} \otimes D_1), \tag{3.13}$$

where $\oplus$ stands for the Kronecker sum. Here the $m_s \times m_s$ matrix $S_{\text{not-all-busy}}$ holds the transition rates across service phases, *between arrivals*, during a not-all-busy period. Since no arrivals are

allowed and the queue is empty, there are no new jobs starting service during this period and the $S_{\text{not-all-busy}}$ matrix holds the same transition rates as the matrix S. The only difference is in the diagonal of $S_{\text{not-all-busy}}$, which needs to be such that $S_{\text{not-all-busy}}\mathbf{1} = \mathbf{0}$. Eq. (3.13) considers that, during an all-busy period, the age reaches a value u with density $\boldsymbol{\pi}(0)\exp(Tu)$. At this point a downward jump of size larger than u occurs with density $A^{(\text{jump})} \otimes \exp(D_0 u)$. This results in a transition to a not-all-busy period, where the phase process evolves according to the sub-generator $S_{\text{not-all-busy}} \oplus D_0$, until an arrival occurs and triggers the process into a new all-busy period. Notice that when a job arrives during a not-all-busy period, the service phase $S_N(t)$ remains unchanged since every queue receives a new request, keeping their differences unchanged. The distribution of the phase just after the arrival that initiates the all-busy period, given the phase just before this arrival, is thus $(I_{m_s} \otimes D_1)$.

To find $\boldsymbol{\pi}(0)$, we rely on the integral

$$L = \int_0^\infty \exp(Tt)(I_{m_s} \otimes \exp(D_0 t))dt$$

defined in Eq. (2.10), which simplifies Eq. (3.13) to

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0)L(A^{(\text{jump})} \otimes I_{m_a})(S_{\text{not-all-busy}} \oplus D_0)^{-1}(I_{m_s} \otimes D_1).$$

Here $\boldsymbol{\pi}(0)$ can be found readily in $O(m^3)$ time, once L has been obtained.

After obtaining matrix T and vector $\boldsymbol{\pi}(0)$, we readily have the waiting time distribution from Eq. (2.18). Recall that the probability that a job has to wait is $\gamma = (E[\eta_0]-1)/(E[\eta_0]-1+E[\eta_1])$. Here η_0 is the number of service completions in an all-busy period, and η_1 is the number of arrivals in a not-all-busy period. The expected value of $E[\eta_0]$ is given by Eq. (2.19), and $E[\eta_1] = 1$ as an arrival during the not-all-busy period sends replicas to all queues in the node, initiating an all-busy period.

Table 3.4: Transition rates for $\bar{S}_{\text{ser}}$, $\bar{\boldsymbol{S}}^*_S$ and $\bar{\boldsymbol{S}}^*_F$

From	To	Rate	Range
$l, (d_1, \ldots, d_{r-1})$	$l{-}1, (\max\{C, d_1{+}1\}, \ldots, d_{r-1})$	α	$l = r$
$l, (d_1, \ldots, d_{r-1})$	S	μ	$l = r$
$l, (0, \ldots, 0, d_i, \ldots, d_{r-1})$	$l{-}1, (1, 0, \ldots, 0, d_i, \ldots, d_{r-1})$	$i\alpha$	$l{=}r, d_i{>}0, d_j{=}0 \quad \forall j{<}i$
$l, (0, \ldots, 0, d_i, \ldots, d_{r-1})$	S	$i\mu$	$l{=}r, d_i{>}0, d_j{=}0 \quad \forall j{<}i$
$l, (\ldots, d_i, d_{i+1}, \ldots, d_{r-1})$	$l, (\ldots, d_i{-}1, d_{i+1}, \ldots, d_{r-1})$	μ	$d_i{>}0,\ d_{i+1}{>}0,\ l{\leq}r$
$l, (\ldots, d_i, d_{i+1}, \ldots, d_{r-1})$	$l, (\ldots, d_i{-}1, \max\{C, d_{i+1}{+}1\}, \ldots, d_{r-1})$	α	$d_i{>}0,\ d_{i+1}{>}0,\ l{\leq}r$
$l, (\ldots, d_i, 0, \ldots, 0, d_k, \ldots, d_{r-1})$	$l, (\ldots, d_i{-}1, 0, \ldots, 0, d_k, \ldots, d_{r-1})$	$(k{-}i)\mu$	$d_i{>}0,\ d_j{=}0\ \forall i{<}j{<}k,\ l{\leq}r$
$l, (\ldots, d_i, 0, \ldots, 0, d_k, \ldots, d_{r-1})$	$l, (\ldots, d_i{-}1, 1, 0, \ldots, 0, d_k, \ldots, d_{r-1})$	$(k{-}i)\alpha$	$d_i{>}0,\ d_j{=}0\ \forall i{<}j{<}k,\ l{\leq}r$
$l, (\mathbf{0}_{r-l-1}, 1, d_{r-l+1}, \ldots, d_{r-1})$	$l{-}1, (\mathbf{0}_{r-l-1}, 0, \max\{C, d_{r-l+1}{+}1\} \ldots, d_{r-1})$	α	$d_{r-l+1}{>}0,\ 1 < l < r$
$l, (\mathbf{0}_{r-l-1}, 1, d_{r-l+1}, \ldots, d_{r-1})$	S	μ	$d_{r-l+1}{>}0,\ 1 < l < r$
$l, (\mathbf{0}_{r-l-1}, 1, 0, \ldots, 0, d_k, \ldots, d_{r-1})$	$l{-}1, (\mathbf{0}_{r-l-1}, 0, 1, 0, \ldots, 0, d_k, \ldots, d_{r-1})$	$(k{-}r{+}l{+}1)\alpha$	$1 < l < r$
$l, (\mathbf{0}_{r-l-1}, 1, 0, \ldots, 0, d_k, \ldots, d_{r-1})$	S	$(k{-}r{+}l{+}1)\mu$	$1 < l < r$
$1, (\mathbf{0}_{r-2}, 1)$	S	μ	$l = 1$
$1, (\mathbf{0}_{r-2}, 1)$	F	α	$l = 1$

3.3.2 The service-time distribution

We now determine the job service-time distribution, which we show to be PH with parameters $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ that depend on the overall system state. Let $Y(t)$ be the service state of a *tagged job* in service at time t. We define $Y(t) = (R(t), S(t))$, where $R(t)$ records the number of alive replicas of the tagged job at time t, thus $R(t) \in \{1, \ldots, r\}$. The variable $S(t)$ is again the difference between queue lengths, but it focuses on the queue lengths *in front of the tagged replicas only*, ignoring any jobs that arrive after the tagged job. Thus if a tagged replica has already failed, the corresponding queue length is kept at 0. Notice that since we are interested in the service time of a tagged job, we focus on phases where *at least one tagged replica* is in service, and we order the service states according to $R(t)$ in descending order. Further, when $R(t) = 1$, only one tagged replica remains alive, thus it is enough to keep track of the length of the queue where this replica is located. To build the PH representation of the service-time distribution we consider the sub-generator $\bar{S}_{\text{ser}}$, and two absorbing states, S and F, representing the cases where the job completes service successfully or encounters a failure, respectively. We can then write the generator of the service time process (ignoring the zero rows corresponding to the absorbing states) as

$$\begin{bmatrix} \bar{S}_{\text{ser}} & \bar{\boldsymbol{S}}^*_S & \bar{\boldsymbol{S}}^*_F \end{bmatrix},$$

where the absorption vectors $\bar{\boldsymbol{S}}^*_S$ and $\bar{\boldsymbol{S}}^*_F$ hold the absorption rates into states S and F, respectively, and $\bar{\boldsymbol{S}}^* = \bar{\boldsymbol{S}}^*_S + \bar{\boldsymbol{S}}^*_F$.

The transitions among service phases depend not only on whether there is a successful service completion or a failure, but also on which queue this happens. The transition rates of the service process are shown in Table 3.4, which we split in four sets for clarity. The first set of rows considers the case where $R(t) = r$, i.e., none of the tagged replicas has completed service or failed, thus there is at least one tagged replica in service in one of the shortest queues. In the first two rows there is only one tagged replica in service, thus its failure, with rate α, leads d_1 to increase by 1 but bounded to C, while its service completion, with rate μ, marks the service completion of the whole job, thus entering the S state. Rows 3-4 cover a similar case, but with i tagged replicas concurrently in service. In the second set, we consider transitions in queues other than the shortest ones, which do not hold tagged replicas. The first and second rows consider the case of a service completion or a failure in the $(i{+}1)^{\text{th}}$ queue, respectively, assuming there is a single queue with this queue length ($d_{i+1} > 0$). The third and fourth rows consider a similar transition, but in this case the $(i{+}1)^{\text{th}}$ to the k^{th} queues have the same length, thus the transition rates are $(k-i)\mu$ and $(k-i)\alpha$, respectively. In the third set we assume that the number of tagged replicas alive is $1 < l < r$, thus at least one tagged replica has already failed. In fact, the zero vector $\mathbf{0}_{r-l+1}$ in front of the state description corresponds to the queue-length differences among the $r-l$ servers where the tagged replicas have already failed. Here the first two rows correspond to the failure or service completion, respectively, of a tagged replica in the (only) shortest non-zero queue, thus assuming $d_{r-l+1} > 0$. The third and fourth rows cover the same case, but considering multiple (i.e., $k-r+l-1$) shortest queues. Finally, the last set considers the case where only one tagged replica remains in service, thus its service completion marks a successful completion of the whole job, while its failure leads to the job failure.

Having obtained $\bar{S}_{\text{ser}}$, we define $S_{\text{ser}} = \bar{S}_{\text{ser}} \otimes I_{m_a}$, which is a matrix of size m_{ser}. Since the service phase space is ordered in decreasing order according to $R(t)$, we let m_r be the number of phases where $R(t) = r$ and m_0 the remaining phases, such that $m_{\text{ser}} = m_r + m_0$. We can now determine the initial probability vector $\boldsymbol{s}_{\text{ser}}$, which is the stationary probability with which a job starts service in each of the phases.

Proposition 3. *The job service-time follows a PH distribution with parameters* $(\boldsymbol{s}_{ser}, S_{ser})$,

where

$$\boldsymbol{s}_{ser} = [(1-\gamma)\boldsymbol{\pi}(0) + \gamma c\boldsymbol{\alpha}_{busy}(T - S \otimes I_{m_a}) \quad \mathbf{0}_{m_0}]. \tag{3.14}$$

Here $c^{-1} = \boldsymbol{\alpha}_{busy}(T - S \otimes I_{m_a})\mathbf{1}$*, and the zero vector correspond to phases with* $R(t) < r$ *as a job can only start service in phases with* $R(t) = r$*, since it starts with all its replicas alive.*

Proof. Notice that a job can only start service in the m_s service phases with $R(t) = r$, since in other phases one or more tasks have already completed service, thus the $\mathbf{0}_{m_0}$ vector in $\boldsymbol{s}_{\text{ser}}$ corresponds to phases with $R(t) < r$. If a job finds at least one idle server, which occurs with probability $1 - \gamma$, it initiates an all-busy period, and starts service with phase according to $\boldsymbol{\pi}(0)$. On the other hand, a job finds all the servers busy with probability γ, and its starting phase is given by the distribution of the phase in the $(X(t), J(t))$ process after a downward jump, as these are the times when new jobs start service. We know that the joint distribution of the age and the phase is $\boldsymbol{\pi}(x)$. Further, thanks to the level-homogeneity of the process, given that the age is x and the phase is i, the probability that the age reaches $[x + u, x + u + du)$ is independent of x. Thus, the probability that level x is visited after a downward jump, and that this occurs by visiting phase j is given by the j^{th} entry of

$$\begin{aligned} c\int_0^\infty \int_0^\infty \boldsymbol{\pi}(x)\exp(Tu)(A^{(\text{jump})} \otimes \exp(D_0 u)D_1)dudx &= c\boldsymbol{\pi}(0)(-T)^{-1}(T - S \otimes I_{m_a}) \\ &= c\boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a}), \end{aligned}$$

where the inner integral is equal to $T - S \otimes I_{m_a}$ from Eq. (2.9), and c is a normalizing constant that is therefore given by $c^{-1} = \boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a})\mathbf{1}$. Thus, a job that waits starts service according to $\boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a}) / (\boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a})\mathbf{1})$. Multiplying this vector by the probability that a job has to wait γ, and adding it to the $(1-\gamma)\boldsymbol{\pi}(0)$ term completes the result. □

As a result, we have that $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ is a PH representation of the service time for all jobs. For further reference we also define the corresponding exit vector $\boldsymbol{S}^*_S = \bar{\boldsymbol{S}}^*_S \otimes \mathbf{1}_{m_a}$. However, we want to focus on the successful jobs only, for which we obtain the PH representation of the

service- and response-time distributions in Proposition 4 and Theorem 3.1, respectively. The proofs, which were done by Dr. Juan F. Pérez, can be found in Appendix A.1.

Proposition 4. *The service time of* successful jobs *follows a PH distribution with parameters* $(\boldsymbol{\beta}_{ser}, B_{ser})$*, where*

$$\boldsymbol{\beta}_{ser} = \boldsymbol{S}_S^{*\prime}\Pi/p_S, \quad B_{ser} = \Pi^{-1}S'_{ser}\Pi.$$

Π *is a diagonal matrix such that* $\Pi\mathbf{1} = \eta'$*,* $\eta = -\boldsymbol{s}_{ser}S_{ser}^{-1}$ *is the stationary distribution of the service phase, and* $p_S = -\boldsymbol{s}_{ser}S_{ser}^{-1}\boldsymbol{S}_S^*$ *is the probability that a job is successful.*

Theorem 3.1. *The response time of successful jobs follows a PH distribution with parameters* $(\boldsymbol{s}_{res}, S_{res})$*, where*

$$\boldsymbol{s}_{res} = \begin{bmatrix} \boldsymbol{\beta}_{ser}^{idle} & \boldsymbol{\beta}_{ser}^{busy} & \mathbf{0} \end{bmatrix}, \; S_{res} = \begin{bmatrix} B_{ser}^{idle} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & B_{ser}^{busy} & (-B_{ser}^{busy}\mathbf{1})P_{s,w} \\ \mathbf{0} & \mathbf{0} & S_{wait} \end{bmatrix}, \tag{3.15}$$

where $(\boldsymbol{\beta}_{ser}^{idle}, B_{ser}^{idle})$ *and* $(\boldsymbol{\beta}_{ser}^{busy}, B_{ser}^{busy})$ *are the PH representations of the service times of successful jobs that start service immediately upon arrival and that must wait, respectively. Letting* Δ_{idle} *be a diagonal matrix such that* $\Delta_{idle}\mathbf{1} = -(1-\gamma)(\boldsymbol{\pi}(0)S_{ser}^{-1})'$*,* $(\boldsymbol{\beta}_{ser}^{idle}, B_{ser}^{idle})$ *are given by*

$$\boldsymbol{\beta}_{ser}^{idle} = \boldsymbol{S}_S^*\Delta_{idle}/p_S, \quad B_{ser}^{idle} = \Delta_{idle}^{-1}S'_{ser}\Delta_{idle}.$$

Similarly, letting Δ_{busy} *be a diagonal matrix such that* $\Delta_{busy}\mathbf{1}$
$=-\gamma(\boldsymbol{\alpha}_{busy}S_{ser}^{-1})'$*,* $(\boldsymbol{\beta}_{ser}^{busy}, B_{ser}^{busy})$ *are given by*

$$\boldsymbol{\beta}_{ser}^{busy} = \boldsymbol{S}_S^*\Delta_{busy}/p_S, \quad B_{ser}^{busy} = \Delta_{busy}^{-1}S'_{ser}\Delta_{busy}.$$

Finally, $P_{s,w}$ *is an* $m_{ser} \times m$ *matrix given by*

$$P_{s,w} = \begin{bmatrix} \tilde{P}_{s,w} \\ 0_{m_0 \times m} \end{bmatrix},$$

3.3.3 Experimental validation

We now demonstrate that the proposed model is able to provide accurate results with quite moderate values of the limit C. To study its behavior, we focus on the mean response time, and the p^{th} response-time percentile, denoted as R_p, which is the maximum response time faced by $p\%$ of the successful jobs. With these two metrics, Table 3.5 summarizes the relative errors of the values obtained from the approximated model (finite C), against simulation results. Similar as Section 3.2, we implemented an event-based simulation in Java. While job service and failure times are generated according to the exponential distribution, we consider Poisson arrivals, and MAPs of second order job inter-arrival times, thanks to the flexibility offered by the proposed model. While the Poisson case is a standard assumption, MAPs allow us to explore the effect of the variability and auto-correlation of the inter-arrival times on the accuracy of the approximation. The variability, measured by the squared coefficient of variation (SCV), is defined for a random variable X as $C_X^2 = \text{Var}[X]/E[X]^2$. While for Exp arrivals the SCV is one, for the MAP case we set the SCV to 10. For MAP arrivals we also set the decay rate of the auto-correlation function to 0.9. We use the method in [HHG06] to obtain the MAP representations. We show the errors achieved when $r = 3$ as an example, and consider different system settings. The reliability of the system without replication is given by $\mu/(\mu + \alpha)\%$, where the service rate μ is set to be 1.0, and the failure rate α is set to achieve different baseline reliability levels, namely 10%, 50% and 90%. Further, the arrival rate is set to achieve different baseline utilization levels for the system without replication. For each configuration we consider different values of the limit C: 5, 10 and 50. For each setting, the simulations were run for 5,000 times with 500,000 samples each time, from which we obtain the mean response time and R_{95}. Focusing on the cases with $r = 3$ and the baseline reliability of 50% and 90%, we observe that the relative error is below 1% for most test cases with C as small as 5, with the exception of the case with MAP arrivals and 0.9 utilization, which requires a larger C of 10 to achieve an error below 1%. Here both the utilization and the more variable and auto-correlated arrival process increase the likelihood of larger queues and larger differences among queue lengths, thus requiring a larger limit C. However, when the reliability is just 10%, we observe significant errors for several cases with $C = 5$ and 10. For instance, with MAP arrivals

Table 3.5: Approximation error (%) compared with simulation results

Arr	Utilization	Measure	Err - reliability:10%			Err - reliability:50%			Err - reliability:90%		
			C = 5	C = 10	C = 50	C = 5	C = 10	C = 50	C = 5	C = 10	C = 50
Exp	0.1	mean	<1	<1	<1	<1	<1	<1	<1	<1	<1
		R_{95}	<1	<1	<1	<1	<1	<1	<1	<1	<1
	0.5	mean	<1	<1	<1	<1	<1	<1	<1	<1	<1
		R_{95}	<1	<1	<1	<1	<1	<1	<1	<1	<1
	0.9	mean	15.2	2.6	<1	<1	<1	<1	<1	<1	<1
		R_{95}	19.5	3.5	<1	<1	<1	<1	<1	<1	<1
MAP	0.1	mean	<1	<1	<1	<1	<1	<1	<1	<1	<1
		R_{95}	<1	<1	<1	<1	<1	<1	<1	<1	<1
	0.5	mean	31.6	11.3	<1	<1	<1	<1	<1	<1	<1
		R_{95}	28.6	9.9	<1	<1	<1	<1	<1	<1	<1
	0.9	mean	34.5	12.0	<1	2.0	<1	<1	<1	<1	<1
		R_{95}	35.1	12.8	<1	1.3	<1	<1	<1	<1	<1

and 0.9 utilization, the error rate with $C = 5$ and 10 are 34.48% and 11.95% for the mean, and 35.08% and 12.78% for R_{95}, respectively. The low reliability case is more challenging as the higher likelihood of failures increases the differences among the queue lengths. Increasing the limit C to 50 allows us to obtain errors below 1%. With medium and high reliability levels, failures are less frequent and the queues stay more synchronized, such that a small C is sufficient to cover most of the queue-length differences observed.

Considering the cases with $r = 2$ we observe similar trends, with errors increasing with the utilization and the arrival process variability, and decreasing with the reliability. However, we observe larger errors than with $r = 3$ thus requiring a larger C to achieve the same level of accuracy. This can be explained by noticing that with r replicas and limit C the maximum difference allowed by the model between the shortest and the largest queues is rC. Thus a larger number of replicas allows the model to consider larger queue-length differences, given the same limit C.

3.3.4 Experimental results

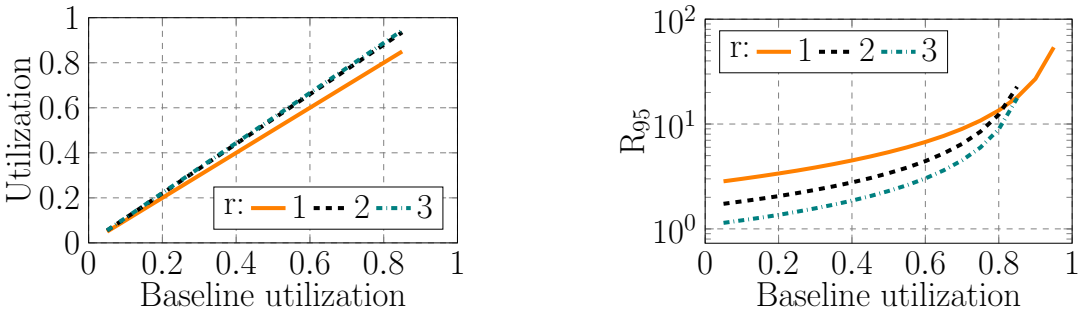
We now make use of the proposed model for the distributed set-up to evaluate its performance in the terms of its reliability and offered response times. Through the whole section, the service rate μ is set to 1, and the mean arrival and failure rates are set in proportion to μ to obtain different *baseline* utilization and reliability (the utilization and reliability achieved without introducing any replicas).

The effect of replication

Beyond the evident improvement in reliability, another benefit of replication with canceling is its potential to reduce the response time by allowing the selection of the first successful result. Focusing on the impact of replication levels has on the response times, Figures 3.2, 3.3 and 3.4 show how the utilization and R_{95} change under different replication levels, under Poisson arrivals, and different baseline reliability levels (10%, 50% and 90%).

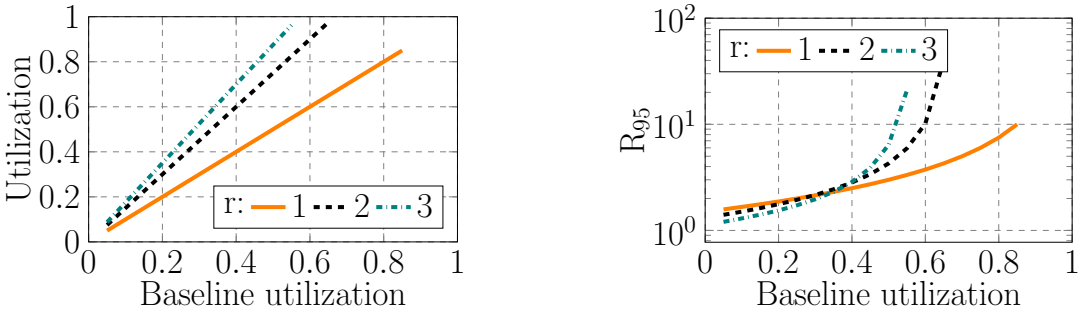
Figure 3.2(a) depicts how the system utilization increases as the replication level increases, although the increase is relatively minor. In this case we assume a reliability of 90%. Decreasing this reliability to 50% and 10%, as depicted in Figures 3.3(a) and 3.4(a), the relative increase in utilization when replication is implemented is larger. For instance, when the reliability is just 10%, a utilization of 0.35 without replication becomes 0.67 when $r = 2$, and further increases to 0.95 when $r = 3$. However, the increase in utilization does not necessarily lead to a higher delay. On the contrary, replication leads to lower response times, as long as the baseline utilization stays below a certain threshold. For instance, Figure 3.2(b) shows that both replication levels achieve a lower R_{95} than without replication, when the baseline utilization is below 0.85. In particular, the system with 3 replicas achieves the lowest response times among the three. Although the introduction of replicas introduces extra load, which leads to an increase in the response times, at the same time it allows the selection of the first replica that finishes, potentially reducing the response times. However, the second effect weakens with a higher failure rate, as we observe in Figures 3.2(b), 3.3(b) and 3.4(b) that the utilization threshold, under which replication is beneficial, decreases with decreasing reliability. In particular, when the baseline reliability is 10%, the introduction of replication increases the response times at any utilization considered. The reason is two-fold: first, the system without replication and a low reliability shows short response times since only short jobs can complete service before a failure; second, the probability that all replicas are running until the first one completes reduces with a higher failure rate, weakening the benefit of selecting the first replica that completes service.

Figure 3.5 shows the utilization and R_{95} obtained under MAP arrivals and a reliability of



(a) Utilization (b) R_{95}

Figure 3.2: Poisson arrivals, 90% baseline reliability



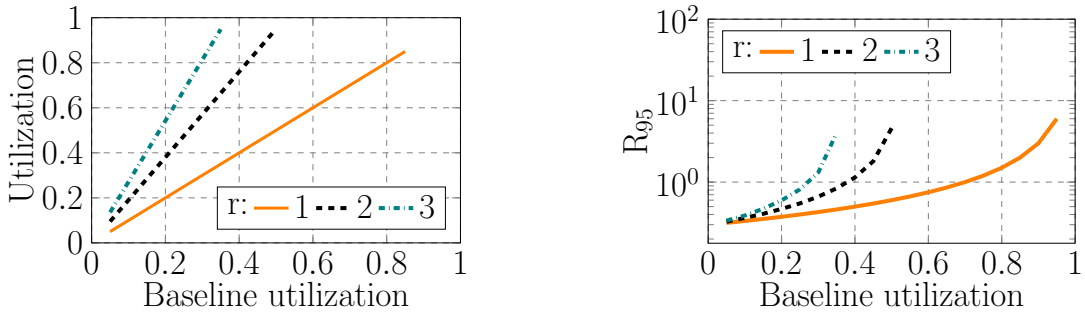
(a) Utilization (b) R_{95}

Figure 3.3: Poisson arrivals, 50% baseline reliability

90%, while keeping the same mean arrival rate as for Poisson arrivals. We observe how the bursty workload modeled by the MAP arrivals leads to much larger response times and a lower utilization threshold, below which replication reduces response times, compared to the case with Poisson arrivals in Figure 3.2. This is caused by the more variable and auto-correlated workload represented by the MAP arrivals, which limit the load range where replication is beneficial.

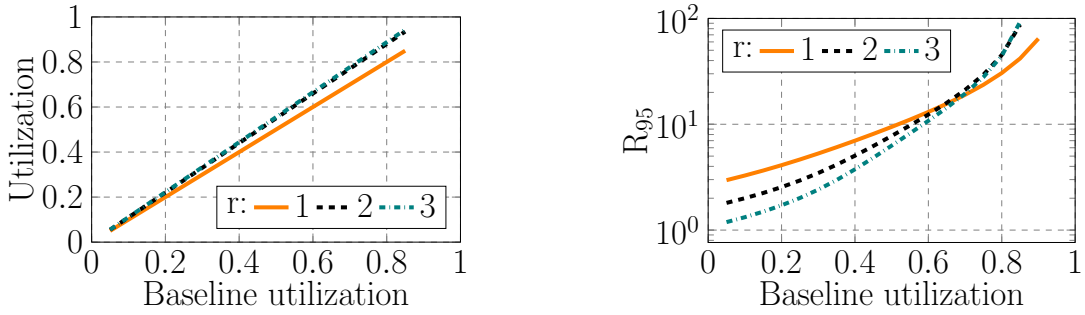
The effect across the distribution

We now look further into the effect that replication has across the whole response-time distribution. To this end, for each percentile p in the range $\{10, 20, \ldots, 90, 99\}$ we obtain the relative improvement $(R_p^1 - R_p^r)/R_p^1\%$, comparing the response time percentiles obtained with replication ($r = 2, 3$) against those without (R_p^1). Figure 3.6(a)-(b) depicts the improvements for the case under Poisson arrivals with reliability of 90% and utilization levels of 0.1 and 0.3, respectively. We observe fairly stable improvements across the whole percentile range, with



(a) Utilization (b) R_{95}

Figure 3.4: Poisson arrivals, 10% baseline reliability

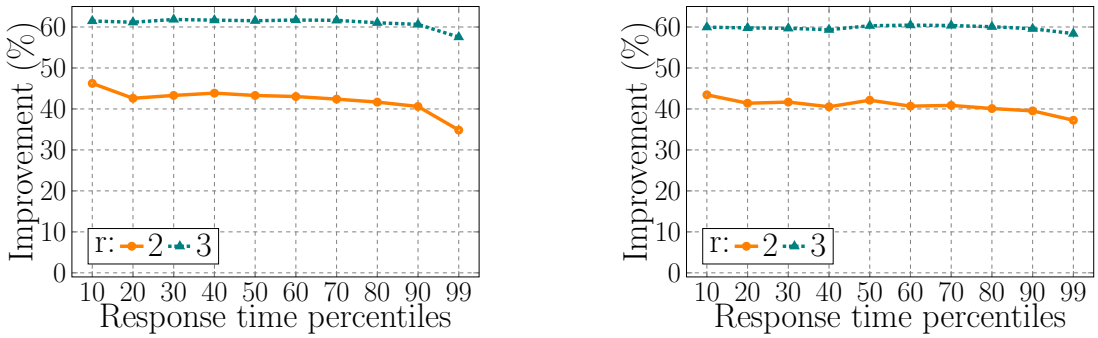


(a) Utilization (b) R_{95}

Figure 3.5: MAP arrivals, 90% baseline reliability

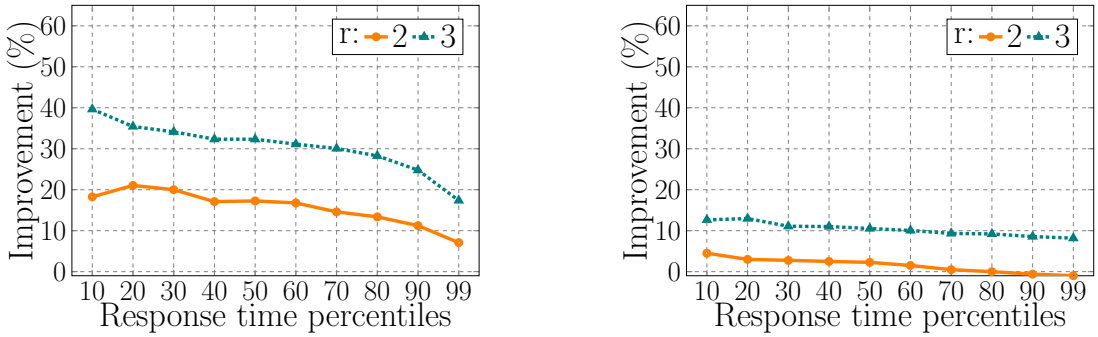
the exception of the improvement on the tail. For instance, when the utilization is 0.1, and $r = 2$, the improvement on the 99$^{\text{th}}$ percentile is 34.83%, while the improvement experienced by most percentiles is around 43%. In fact, we observe that the improvement generally decreases as the percentile considered increases. The reason for this is that a larger percentile covers longer response times, which are partially associated to longer service times. In a long service time there is more room for a failure to occur, in which case the system loses its ability to use the minimum of r execution times. Notice that this effect decreases if r is larger, as shown in Figure 3.6(a)-(b), as even if one replica fails, there are $r-1$ replicas that could execute concurrently and among which the system selects the minimum execution time. Further, this effect is stronger if the utilization is low, as confirmed when comparing Figure 3.6(a) and 3.6(b), since under low utilizations the contribution of the service time on the response time is larger as the queueing times are very short.

In the previous set-up we assumed a reliability of 90%. If we reduce it to 50%, Figure 3.7(a)-(b) shows that the gains across the percentiles are not uniform, where we observe peaks on small



(a) Utilization:0.1 (b) Utilization:0.3

Figure 3.6: Improvement on R_p (Poisson arrivals, reliability: 90%)



(a) Utilization:0.1 (b) Utilization:0.3

Figure 3.7: Improvement on R_p (Poisson arrivals, reliability: 50%)

percentiles, and a stronger decrease in the improvement for larger percentiles. For instance, when $r = 3$ and utilization is 0.1, the improvement of the 10^{th} percentile is 39.64%, while it is 17.36% for the 99^{th} percentile. This more pronounced effect is in agreement with the observation above as in this case failures are more likely and higher percentiles evaluate conditions where service times can be larger and provide more chances for failures to occur. This clearly highlights the importance of explicitly considering the response-time distribution, and not just its mean, or one percentile, when evaluating a replication strategy.

Chapter 4

Fault-tolerance and Latency-reduction via Replication: the Centralized Set-up

In the previous chapter, we have focused on the distributed set-up, where we develop an approximate stochastic model to determine the request response-time distribution. In this chapter, we explore the centralized set-up, where we propose a stochastic model to study how the replication with canceling approach affects the offered response-time *percentiles*, with two replicas in Section 4.1, and general number of replicas in Section 4.2. Finally, in Section 4.3, we compare the distributed and centralized set-ups, where we find that the distributed set-up offers response times close to those of the centralized system, especially at medium-to-high request reliability.

4.1 The case with $r = 2$ replicas

We consider the model as shown in Figure 3.1(c), where all the replicas of the incoming requests form a single centralized queue in the order of arrival and join the next server that becomes available with FCFS scheduling. For example, in Figure 4.1 we show the case with $r = 2$ replicas, where for each request O submitted to the system, a single replica R is adopted and sent right after the original request to the central queue, and the system replies with the result from whichever replica completes first. To analyze the performance of this set-up, we first

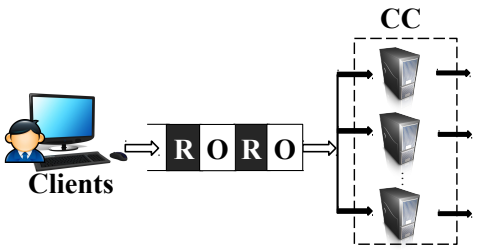


Figure 4.1: Reference model

propose a model to evaluate the performance of a system with $r = 2$ replicas. We then extend the proposed model to cases with $r \geq 2$, limiting the number of replicas to be same as the number of servers in a computing node. This allows the comparison between the distributed and the centralized set-ups. As with the distributed case, while the *request/replica* processing times are exponentially distributed with rate μ, we will show that the *job* processing times follow a PH distribution, whose parameters depend on the overall system state.

4.1.1 The queue-length distribution

In this section we focus on the computation of the queue-length distribution. While relevant on its own, we will make use of the queue-length distribution in Section 4.1.2 to obtain the job response-time distribution. The system without replication can be cast as a MAP/M/c queue with service rate $\mu + \alpha$, and its queue-length distribution can be obtained using the method in [LR99, AM01]. However, the case *with* replication requires a more detailed analysis due to the particular dynamics introduced by both replication and canceling.

For the system with replication, we set up a two-dimensional MC $\{(L(t), J(t))|t \geq 0\}$, where $L(t)$, the *level* variable, holds the number of jobs in the system, and $J(t) = (A(t), S(t))$, the *phase* variable, holds both the phase of the arrival process $A(t)$ and of the service process $S(t)$. The arrival process is a MAP with m_a phases and parameters D_0 and D_1. During a not-all-busy period, we define the service state to be $S(t) = (L_2(t), L_1(t))$, where $L_i(t)$ is the number of jobs with i replicas in service. A job has only one replica in service if the other one already failed. For instance, for a system with 3 servers and 1 job in process, either one or both replicas of this job are in service, leading the possible service phases in level 1 to be $\{(1, 0), (0, 1)\}$. On the other hand, during an all-busy period we define the service state to be $S(t) = (L_2(t), Y(t))$,

where $Y(t)$ represents the state of the youngest job in service. $Y(t)$ takes values from $\{2, O, R\}$, where: 2 means both replicas of the youngest job are in service; O means only the first replica is in service while the second one is waiting in front of the queue; and R means one of the youngest job's replicas failed while the other one is still in service. In this case there is no need to record the number of jobs with one replica in service, since this can be calculated as $c-2L_2(t)$ when all the servers are busy. In the example with 3 servers, but 3 jobs in the system, the possible service phases are $\{(1, O), (1, 2), (1, R), (0, O), (0, R)\}$, where service phase $(1, O)$ represents the case where one job has both replicas in service, the youngest job in service has its first replica in service and its second replica waiting, and one job is waiting in the queue. The number of service phases for level k is given by

$$m_s(k) = \begin{cases} k+1, & k \leq \lfloor c/2 \rfloor, \\ 2k - \lfloor c/2 \rfloor + 1, & \lfloor c/2 \rfloor < k < c. \end{cases} \tag{4.1}$$

From level c onwards, all the levels have the same number of service phases $m_s = 2c - \lfloor c/2 \rfloor$, where $\lfloor a \rfloor$ is the largest integer smaller than or equal to a. We order the state-space lexicographically and define level k as the set of states where k jobs are present, i.e., the set $\{(k, j), 0 \leq j \leq m_k\}$, where $m_k = m_a m_s(k)$, for $0 \leq k < c$, and $m_k = m_a m_s$ for $k \geq c$.

Notice that levels 0 to $\lfloor c/2 \rfloor - 1$ correspond to not-all-busy periods, while levels greater and equal to c correspond to all-busy periods. However, levels $\lfloor c/2 \rfloor$ to $c-1$ have both not-all-busy and all-busy states since a job may have either one of or both replicas in service. The MC therefore has a Quasi-Birth-and-Death (QBD) structure [LR99], as described in Section 2.2.1, with level-dependent transition rates between levels 0 and c, and level-independent rates from level $c+1$. As a result, the infinitesimal generator of the MC can be written as

$$P = \begin{bmatrix} A_1^{(0)} & A_2^{(0)} & 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ A_0^{(1)} & A_1^{(1)} & A_2^{(1)} & 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 & A_0^{(c)} & A_1^{(c)} & A_2^{(c)} & 0 & \cdots \\ 0 & \ddots & \ddots & \ddots & 0 & A_0 & A_1 & A_2 & \cdots \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 & A_0 & A_1 & \cdots \end{bmatrix}.$$

Table 4.1: Transition rates for levels $0 \leq k \leq c$

Matrix	Condition	From	To	Rate
$C_0^{(k)}$	$k_2 > c-k$	$(k_2, 2)$	(k_2-1, O)	$2k_2\alpha$
		(k_2, O)	(k_2, R)	α
		(k_2, O)	$(k_2, 2)$	$2k_2\alpha$
		(k_2, R)	(k_2-1, O)	$2k_2\alpha$
	$k_2 = c-k$	$(k_2, 2)$	$(k_2-1, c-2k_2+1)$	$2k_2\alpha$
		(k_2, O)	(k_2, R)	α
		(k_2, O)	$(k_2, 2)$	$2k_2\alpha$
		(k_2, R)	$(k_2-1, c-2k_2+1)$	$2k_2\alpha$
	$k_2 < c-k$	(k_2, k_1)	(k_2-1, k_1+1)	$2k_2\alpha$
$C_1^{(k)}$	$k_2 > c-k$	$(k_2, 2)$	$(k_2, 2)$	$2k_2\mu$
		$(k_2, 2)$	(k_2, O)	$(c-2k_2)\beta$
		(k_2, O)	(k_2, O)	$(2k_2+1)\mu$
		(k_2, O)	$(k_2+1, 2)$	$(c-2k_2-1)\beta$
		(k_2, R)	$(k_2, 2)$	$2k_2\mu$
		(k_2, R)	(k_2, O)	$(c-2k_2)\beta$
	$k_2 = c-k$	$(k_2, 2)$	$(k_2-1, c-2k_2)$	$2k_2\mu$
		$(k_2, 2)$	$(k_2, c-2k_2-1)$	$(c-2k_2)\beta$
		(k_2, O)	$(k_2, c-2k_2-1)$	$(2k_2+1)\mu$
		(k_2, O)	$(k_2+1, 2)$	$(c-2k_2-1)\beta$
		(k_2, R)	$(k_2-1, c-2k_2)$	$2k_2\mu$
		(k_2, R)	$(k_2, c-2k_2-1)$	$(c-2k_2)\beta$
	$k_2 < c-k$	(k_2, k_1)	(k_2-1, k_1)	$2k_2\mu$
		(k_2, k_1)	(k_2, k_1-1)	$k_1\beta$

The matrices $A_0^{(k)}$, $A_1^{(k)}$, and $A_2^{(k)}$ hold the transition rates from level k to levels $k-1$, k, and $k+1$, respectively. From level $c+1$ onwards, these matrices do not depend on the level and are simply labeled A_0, A_1, and A_2. While a successful service completion of any replica in service leads to a level decrease, a failure may not necessarily lead to the change of the level variable. For example, if one job has both replicas in service, and one of them fails, the number of jobs in the system is not altered as its other replica remains in service. We therefore model the evolution of the service phase in level k as a MAP with parameters $(m_s(k), C_0^{(k)}, C_1^{(k)})$, removing the super-index k for levels greater than or equal to $c+1$ as these transitions become independent of the level. The matrices $C_0^{(k)}$, resp. $C_1^{(k)}$ hold the service transition rates without, resp. with, a change of the level. Letting I_n denote the $n \times n$ identity matrix, the QBD blocks for levels $k \geq c+1$ are given by

$$A_0 = C_1 \otimes I_{m_a}, \quad A_1 = C_0 \otimes I_{m_a} + I_{m_s} \otimes D_0, \quad A_2 = I_{m_s} \otimes D_1,$$

and similarly for levels $0 \leq k \leq c$, adding the super-index k.

Let the MC stationary probability vector $\boldsymbol{\pi}$ be partitioned as $\boldsymbol{\pi} = [\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \dots]$, where the

$1 \times m_k$ vector $\boldsymbol{\pi}_k$ corresponds to level k, and its j-th element $\pi_{k,j}$ holds the probability that k jobs are in the system, while the phase is j. The QBD structure implies that $\boldsymbol{\pi}_{c+i} = \boldsymbol{\pi}_c R^i$, for $i \geq 1$, thus the stationary distribution for levels $k \geq c+1$ can be obtained from $\boldsymbol{\pi}_c$ and R. The matrix R is the minimal non-negative solution to the matrix equation $R = A_2 + RA_1 + R^2A_0$, and can be obtained with, for instance, the Cyclic Reduction algorithm [BLM05]. To determine the sub-vector $[\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_c]$ we modify the transition rates within level c to $A_1^{(c)} + RA_0$, thus censoring the transitions beyond this level. The result is a finite level-dependent QBD, the stationary probability vector of which can be found with the linear level-reduction algorithm in [LR99]. Once the sub-vector $[\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_c]$ is obtained, it must be normalized with the condition $\sum_{k=0}^{c-1} \boldsymbol{\pi}_k \mathbf{1} + \boldsymbol{\pi}_c (I - R)^{-1} \mathbf{1} = 1$. Recall that the vector $\boldsymbol{\pi}$ exists if the QBD MC is positive recurrent, which occurs under the condition Eq. (2.4). We now describe how to determine the matrices $(C_0^{(k)}, C_1^{(k)})$ that govern the evolution of the service process.

1. Levels $0 \leq k \leq c$

For levels 0 to c, each level has a different number of service phases, as in Eq. (4.1), and transitions may lead to a switch between not-all-busy and all-busy periods states. Service transition rates depend on different conditions as shown in Table 4.1, where $\beta = \mu + \alpha$. Three different conditions are possible: when $k_2 > c - k$, there is at least one job waiting in the queue; when $k_2 = c - k$, either the queue is empty or the second replica of the youngest job is in the queue; and when $k_2 < c - k$, at least one server is idle.

2. Levels $k \geq c + 1$

For levels greater than or equal to $c + 1$, the transitions are independent of the level. The transition rates of matrices C_0 and C_1 are summarized in Table 4.2. As an example, consider service phase $(k_2, 2)$ $(k_2 \geq 1)$ in level k, k_2 of the k jobs present have both replicas in service. From this state, either replica of the k_2 jobs may complete service and cancel its partner immediately, leading to service phase $(k_2, 2)$ in level $k - 1$, with rate $2k_2\mu$.

Table 4.2: Transition rates for levels $k \geq c+1$

Matrix	From	To	Rate
C_0	$(k_2, 2)$	(k_2-1, O)	$2k_2\alpha$
	(k_2, O)	$(k_2, 2)$	$2k_2\alpha$
	(k_2, O)	(k_2, R)	α
	(k_2, R)	(k_2-1, O)	$2k_2\alpha$
C_1	$(k_2, 2)$	$(k_2, 2)$	$2k_2\mu$
	$(k_2, 2)$	(k_2, O)	$(c-2k_2)\beta$
	(k_2, O)	(k_2, O)	$(2k_2+1)\mu$
	(k_2, O)	$(k_2+1, 2)$	$(c-2k_2-1)\beta$
	(k_2, R)	$(k_2, 2)$	$2k_2\mu$
	(k_2, R)	(k_2, O)	$(c-2k_2)\beta$

4.1.2 The service-time distribution

We now show that the response-time distribution has a PH representation that is obtained by considering the service-time and waiting-time separately. To obtain the job service-time distribution, we make extensive use of the $\boldsymbol{\pi}$ vector found in the previous section for the queue-length process. In the system with replication, the *job* service time does not follow an exponential distribution since it is composed of two replicas, whose execution depend on the system state. The service time however has a PH representation with parameters $\{\boldsymbol{s}_{\text{ser}}, S_{\text{ser}}\}$. Let $Y(t)$ be the service state of a tagged job at time t, which takes values from $\{2, O, R\}$, as defined in Section 4.1.1. We add two absorbing states S and F to represent the cases where the job completes service successfully or encounters a failure, respectively. Further, the holding times in each state are exponentially distributed, and we can describe the transition among the states as an MC with generator $S_{\text{ser}} = \bar{S}_{\text{ser}} \otimes I_{m_a}$, where $\bar{S}_{\text{ser}}$ is given by

$$[\bar{S}_{\text{ser}}|\bar{\boldsymbol{S}}^*_S|\bar{\boldsymbol{S}}^*_F] = \begin{array}{c} \\ (2) \\ (\text{O}) \\ (\text{R}) \end{array} \begin{array}{c} \begin{array}{ccccccc} (2) & & (\text{O}) & & (\text{R}) & & (\text{S}) \quad\quad (\text{F}) \end{array} \\ \begin{pmatrix} -2\beta & 0 & 2\alpha & \vdots & 2\mu & \vdots & 0 \\ (c-1)\beta & -c\beta & \alpha & \vdots & \mu & \vdots & 0 \\ 0 & 0 & -\beta & \vdots & \mu & \vdots & \alpha \end{pmatrix} \end{array}.$$

We have labeled $\bar{\boldsymbol{S}}^*_S$ and $\bar{\boldsymbol{S}}^*_F$ the absorption vector to states S and F, respectively, while S_{ser} is precisely the sub-generator matrix we need for the PH representation of the service-time distribution. Notice that a job can only start service in phase 2 or O, since in R one of the replicas has already failed. Further, for phases 2 and R we only need to consider the state of

Table 4.3: Transition rates of $Q_S(k)$, $k \geq \lfloor c/2 \rfloor + 1$

Condition	Current state	Initial states	Transition rates
$k_2 > c - k$	$(k_2, 2)$	2	$2k_2\mu$
	$(k_2, 2)$	O	$(c - 2k_2)\beta + 2k_2\alpha$
	(k_2, O)	O	$(2k_2 + 1)\mu$
	(k_2, R)	2	$2k_2\mu$
	(k_2, R)	O	$(c - 2k_2)\beta + 2k_2\alpha$

the tagged job's replicas in process. Instead, in phase O, which only occurs when all the servers are busy, the first replica of the tagged job is in service while its second replica is waiting in front of the queue. Thus, if *any* of the other $c-1$ requests in service completes service or fails, the second replica of the tagged job starts service. This occurs with rate $(c-1)\beta$ and leads the tagged job to service phase 2.

Having obtained S_{ser}, it remains to determine the initial probability vector $\boldsymbol{s}_{\text{ser}}$, which is the stationary probability that a job starts service in each of the phases $\{2, O, R\}$. In other words, $\boldsymbol{s}_{\text{ser}}$ is the distribution of the service phase of the youngest job in service, immediately after the initiation of a job service. This can be obtained from the queue-length process defined in Section 4.1.1 by multiplying the stationary distribution $\boldsymbol{\pi}$ by a matrix that holds the rates at which a new job starts service in each state, and the phase in which the service starts. We can therefore obtain $\boldsymbol{s}_{\text{ser}}$ as

$$\frac{\sum_{k=0}^{\lfloor c/2 \rfloor} \boldsymbol{\pi}_k Q_A(k) + \sum_{k=\lfloor c/2 \rfloor + 1}^{c-1} \boldsymbol{\pi}_k (Q_A(k) + Q_S(k)) + \sum_{k=c}^{\infty} \boldsymbol{\pi}_k Q_S(k)}{\left(\sum_{k=0}^{\lfloor c/2 \rfloor} \boldsymbol{\pi}_k Q_A(k) + \sum_{k=\lfloor c/2 \rfloor + 1}^{c-1} \boldsymbol{\pi}_k (Q_A(k) + Q_S(k)) + \sum_{k=c}^{\infty} \boldsymbol{\pi}_k Q_S(k) \right) \mathbf{1}}, \tag{4.2}$$

where the denominator is simply the sum of the numerator entries. The matrices $Q_A(k)$ and $Q_S(k)$ hold the rates at which a new job starts service due to an arrival or to a service completion in level k, respectively, as described next.

The matrices $Q_A(k)$ depend on the level k and are defined up to level $c-1$, as beyond this level an arrival does not trigger a job service start. These matrices can be written as $Q_A(k) = \bar{Q}_A(k) \otimes D_1$, where the (i, j)-th entry of $\bar{Q}_A(k)$ is equal to 1 if the arrival of a job to a system with k jobs and service phase i causes the start of a new job in service state j; otherwise it is equal to 0.

Table 4.4: Transition rates of S and $A^{(\text{jump})}$

Matrix	From	To	Rate
S	(k_2, O)	(k_2, R)	α
	(k_2, O)	$(k_2+1, 2)$	$(c-2k_2-1)\beta$
	(k_2, O)	$(k_2, 2)$	$2k_2\alpha$
$A^{(\text{jump})}$	(k_2, O)	(k_2, O)	$(2k_2+1)\mu$
	$(k_2, 2)$	$(k_2, 2)$	$2k_2\mu$
	$(k_2, 2)$	(k_2-1, O)	$2k_2\alpha$
	$(k_2, 2)$	(k_2, O)	$(c-2k_2)\beta$
	(k_2, R)	$(k_2, 2)$	$2k_2\mu$
	(k_2, R)	(k_2, O)	$(c-2k_2)\beta$
	(k_2, R)	(k_2-1, O)	$2k_2\alpha$

The matrices $Q_S(k)$ are only relevant for states where all the servers are busy and at least one job is waiting in the queue, i.e., for states in levels $k \geq \lfloor c/2 \rfloor + 1$ satisfying the condition $k_2 > c-k$. Only in this case a service completion or failure may trigger the first job in the queue to start service. Further, for levels $c+1$ or greater we remove the index k as these matrices no longer depend on the level k. We let $Q_S(k) = \bar{Q}_S(k) \otimes I_{m_a}$, where the (i,j)-th entry of $\bar{Q}_S(k)$ holds the service and failure rate in service phase i, for $1 \leq i \leq m_s(k)$, that triggers the start of a job in service phase j. The transition rates of $Q_S(k)$ are shown in Table 4.3.

4.1.3 The waiting-time distribution

To determine the waiting-time distribution, we follow the process described in Section 3.3.1, observing the queue *only during the all-busy periods* [AM01], and defining a bivariate Markov process $\{X(t), J(t) | t \geq 0\}$ [Sen89]. Here the age process $X(t)$ is again the total time-in-system of the youngest *job* in service, and the phase process $J(t) = (A(t), L_2(t), Y(t))$ holds the state of the arrival and service processes, as in the queue-length process defined in Section 4.1.1. The phase process $J(t)$ thus takes $m = m_a m_s$ different values. Different from the standard case treated in [AM01], where the service completion of any job in service triggers the start of a new job service, we need to keep track of the service state of the youngest job in service $Y(t)$ as part of the phase process. This is necessary because, a service completion or failure does not necessarily trigger the start of a new job service, as a replica of the youngest job may be the next to start service. For instance, if $Y(t) = O$, a service completion or failure leads the second replica of the youngest job to start service, *without a new job* stating service.

To determine the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution, we rely on the stationary distribution $\boldsymbol{\pi}(x)$ of the $(X(t), J(t))$ process, which has a matrix exponential representation $\boldsymbol{\pi}(x) = \boldsymbol{\pi}(0)\exp(Tx)$ as in Eq. (2.8), thus involving to find the matrix T first. To compute matrix T, we rely on Eq. (2.9), which requires to determine matrices S and $A^{(\text{jump})}$ first. Recall that matrices S and $A^{(\text{jump})}$ hold the transition rates of the service process associated to transitions without and with the start of a new job service, respectively, as described in Section 2.5. Table 4.4 summarizes their transition rates. The vector $\boldsymbol{\pi}(0)$ can be obtained by observing that its i-th element equals the probability that the phase $J(t)$ is i immediately after an arrival that finds $c-2$ or $c-1$ busy servers, thus starting an all-busy period. We thus define the $1 \times m_k$ vectors $\boldsymbol{u}_k$ that hold the number of busy servers in each state of level k. Further, let the i-th entry of the $1 \times m_k$ vector $\boldsymbol{v}_k$ be $[\boldsymbol{v}_k]_i = \mathbf{1}\{u_{k,i} = c-1|c-2\}$, where the indicator function $\mathbf{1}\{x\}$ equals 1 if x is true and 0 otherwise. Therefore, $\boldsymbol{\pi}(0)$ can be computed as

$$\boldsymbol{\pi}(0) = \frac{\left(\sum_{k=0}^{c-1}(\boldsymbol{\pi}_k \circ \boldsymbol{v}_k)(K_k \otimes I_{m_a})\right)(I_{m_s} \otimes D_1)}{\left(\sum_{k=0}^{c-1}(\boldsymbol{\pi}_k \circ \boldsymbol{v}_k)(K_k \otimes I_{m_a})\right)(I_{m_s} \otimes D_1)\mathbf{1}},$$

where $\circ$ stands for the Hadamard product [Hor90], and the $m_s(k) \times m_s$ matrix K_k holds the first $m_s(k)$ rows of I_{m_s}.

Similar, the probability γ that all servers are busy at a job's arrival instant can be obtained by relying on the queue-length process as

$$\gamma = 1 - \frac{\sum_{k=0}^{c-1}(\boldsymbol{\pi}_k \circ \boldsymbol{w}_k)(I_{m_s(k)} \otimes D_1)\boldsymbol{e}_{m_s(k)}}{\sum_{k=0}^{\infty}(\boldsymbol{\pi}_k(I_{m_s(k)} \otimes D_1)\boldsymbol{e}_{m_s(k)})},$$

where the entries of $\boldsymbol{w}_k$ are given by $[\boldsymbol{w}_k]_i = \mathbf{1}\{u_{k,i} \leq c-1\}$, identifying the states with at least one idle server.

To this end, we follow the process described in Section 2.5 to find the waiting-time distribution $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$. Given the PH representations of both waiting and service times, the PH representation $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$ of the response-time distribution is given by Eq. (2.21). From this, we readily have the response-time PDF, CDF, and moments. The average number of busy servers U can

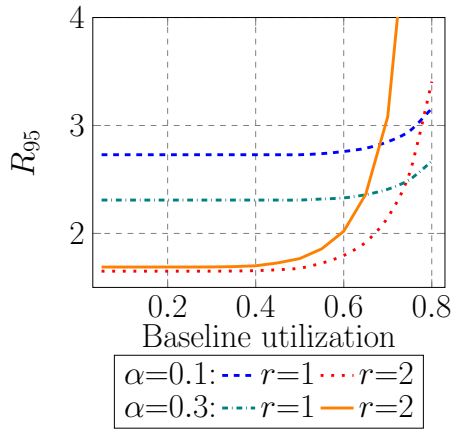


Figure 4.2: The failure rate effect

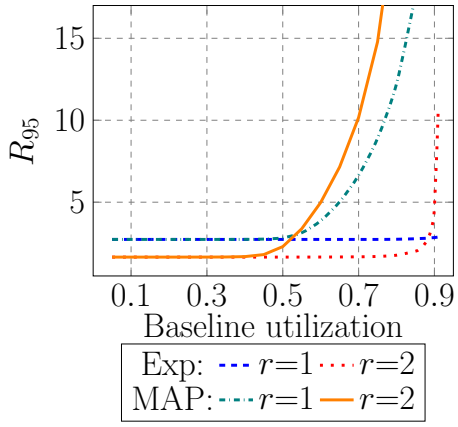


Figure 4.3: The IAT process effect

be computed as

$$U = \left(\sum_{k=1}^{c-1} k\boldsymbol{\pi}_k \circ \boldsymbol{u}_k + c \left(1 - \sum_{k=1}^{c-1} \boldsymbol{\pi}_k \mathbf{1} \right) \right) / c.$$

4.1.4 Experimental assessment

In this section, we make use of the proposed model to evaluate the impact of replication has on the system performance. To illustrate the impact of the arrival processes, we consider renewal processes with exponential (Exp) and 2-phase hyper-exponential (HE_2) inter-arrival times (IATs), as well as 2-phase MAPs (MAP). The squared coefficient of variation, as defined in Section 3.3.3 as SCV, is set to 1 for Exp arrivals, and 10 for the HE_2 and MAP cases. For MAP arrivals we also set the decay rate of the auto-correlation function to 0.9. We use the methods in [Whi82, HHG06] to obtain the HE_2 and MAP representations. The service rate μ is again set to 1, and the arrival rate λ and failure rate α are in proportion to μ.

In additional to the improvement on the reliability, another benefit of replication is its ability to exploit the concurrent execution of the replicas to select the first available result. We have observed that this effect can effectively reduce the response times in the distributed set-up, as long as the utilization of the system without replication stays below a certain threshold. To study this behavior in the centralized set-up, we focus on the response time p-th percentile R_p. Figure 4.2 depicts R_{95} as a function of the baseline utilization (the utilization of the system

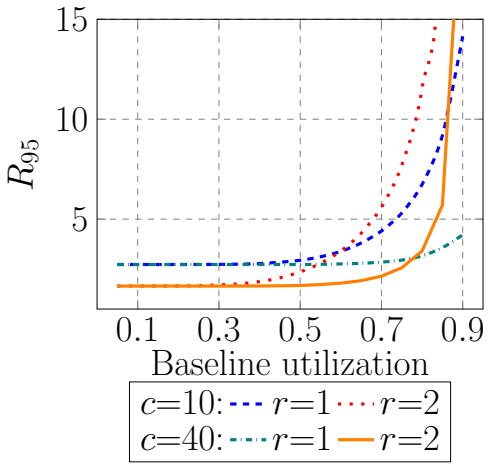


Figure 4.4: The number of servers

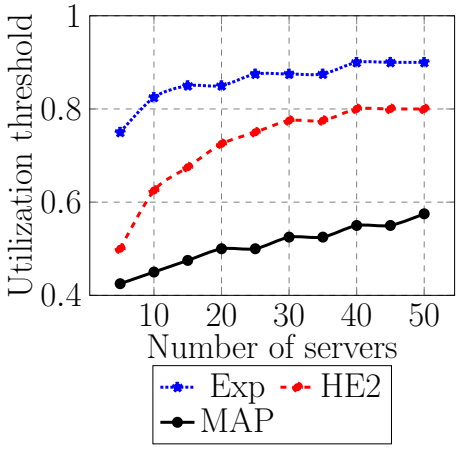


Figure 4.5: Utilization threshold

without replication), under HE_2 arrivals, 40 servers, and failure rates 0.1 and 0.3. Focusing on the case with failure rate 0.1, we observe how R_{95} is 40% smaller under replication, when the utilization is up to 40%. For larger utilization levels, the difference reduces and becomes zero at a utilization threshold of 0.78. Thus, below this value, the introduction of replication effectively reduces the response times. Figure 4.2 also captures the effect of the failure rate on the response-time distribution. Without replication, the system with *higher* failure rate shows a shorter R_{95}, as with a lower reliability, only short jobs can complete service before a failure. However, the introduction of replication makes the R_{95} under failure rate 0.1 to be shorter than under failure rate 0.3. This is caused by the ability of the replication strategy to obtain the result of the first replica to finish, but this effect weakens with a higher failure rate, as this reduces the probability that both replicas are running until the first one completes. Further, the threshold diminishes under the higher failure rate, limiting the conditions under which replication reduces the response times.

Figure 4.3 compares the R_{95} under Exp and MAP arrivals, keeping the same mean arrival rate. First, we observe how the bursty workload modeled by the MAP arrivals has a large effect on the response times, although this effect only takes shape after a certain utilization threshold, in this case between 40% and 50%. Same as the distributed case, the system under MAP arrivals has a smaller threshold than under Exp arrivals, and we observe how under MAP arrivals and replication, the response times start to grow earlier than without replication.

Figure 4.4 compares the R_{95} under 10 and 40 servers, with HE_2 arrivals. The utilization

Table 4.5: Transition rates of matrix S

From		To		Rate	Condition
$(l_r(t),\ldots,l_1(t))$	$Y(t)$	$(l_r(t),\ldots,l_1(t))$	$Y(t)$		
$(l_r,\ldots,l_i,\ldots,l_1)$	(i,j)	$(l_r,\ldots,l_{i+1}{+}1,l_i{-}1,\ldots,l_1{-}1)$	$(i{+}1,j{-}1)$	$l_1\beta$	$j{\geq}1$
$(l_r,\ldots,l_b,\ldots,l_1)$	(i,j)	$(l_r,\ldots,l_b{-}1,\ldots,l_i{-}1,\ldots,l_{i+b}{+}1,\ldots,l_1)$	$(i{+}b,j{-}b)$	$bl_b\mu$	$1{<}b{\leq}r,j{\geq}b$
$(l_r,\ldots,l_b,\ldots,l_1)$	(i,j)	$(l_r,\ldots,l_b{-}1,l_{b-1}{+}1,\ldots,l_{i+1}{+}1,l_i{-}1,\ldots,l_1)$	$(i{+}1,j{-}1)$	$bl_b\alpha$	$1{<}b{\leq}r,j{\geq}1$
$(l_r,\ldots,l_i,\ldots,l_1)$	(i,j)	$(l_r,\ldots,l_i,\ldots,l_1)$	$(i,j{-}1)$	$i\alpha$	$1{\leq}i{\leq}r,j{\geq}1$

threshold the 10-server system is smaller than that of the 40-servers case, as more servers provide more flexibility to handle the additional load introduced with the replicas. We also observe how the R_{95} is the same for the 10 and 40 server cases up to a certain utilization level, after which they start to diverge. This is because under low utilizations, response times mostly correspond to service times, and as the utilization increases, queueing times become more important. We conclude this section with Figure 4.5, which shows the threshold for Exp, HE_2, and MAP arrivals, with the number of servers between 5 to 50. This figure summarizes how the variability and auto-correlation of the arrival process affect the replication decisions, and how additional servers provide more flexibility to deal with the additional load.

4.2 The case with $r \geq 2$ replicas

Having obtained the response-time distribution for jobs in the centralized set-up with $r = 2$ replicas and $c \geq 1$ servers, we now extend the model to consider the general cases with $r \geq 2$. However, due to the complexity of the problem, the model becomes numerically intractable for cases with $r \geq 2$ and general number of servers c, thus we restrict $c = r$. This also enables the comparison between the centralized and the distributed set-up.

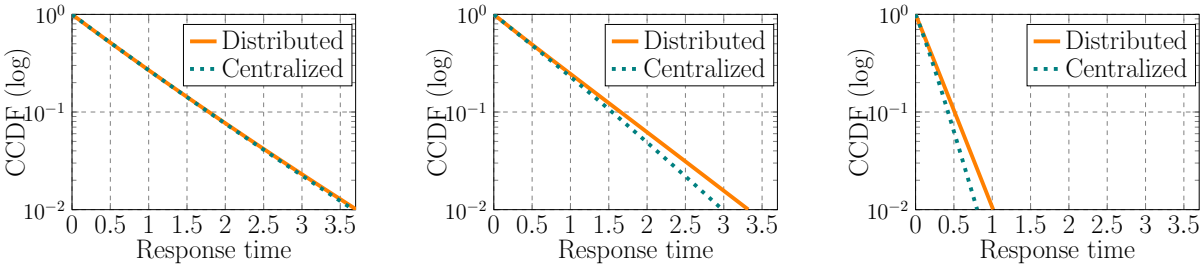
4.2.1 The waiting-time distribution

Similar to the $r = 2$ case, we define a bivariate Markov process $\{X(t), J(t)|t \geq 0\}$. However, the phase $J(t)$ is defined as $J(t) = (l_r(t),\ldots,l_1(t),Y(t))$, where $l_i(t)$ is the number of jobs with i replicas in service for $1 \leq i \leq r$, and $Y(t)$ holds the state of the youngest job in service. Thus, $Y(t) = (i,j)$ means that i replicas of the youngest job are in service, j replicas are

Table 4.6: Transition rates of matrix $A^{(\text{jump})}$

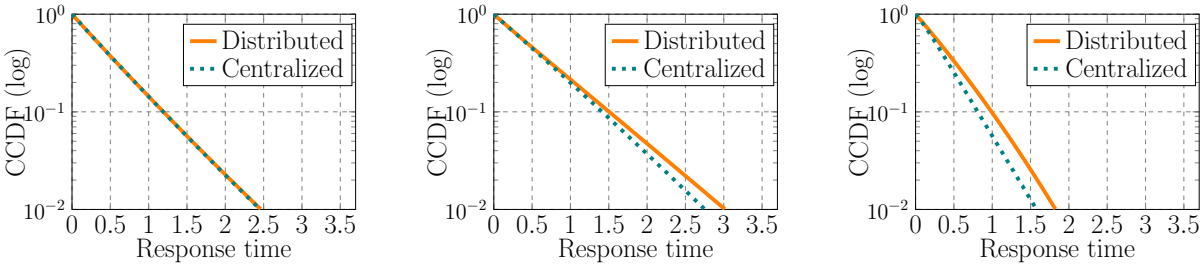
From		To		Rate	Condition
$(l_r(t),\ldots,l_1(t))$	$Y(t)$	$(l_r(t),\ldots,l_1(t))$	$Y(t)$		
$(l_r,\ldots,l_i,\ldots,l_1)$	(i,j)	$(l_r,\ldots,l_i,\ldots,l_1)$	$(i,r-i)$	$i\mu$	$1\leq i\leq r$
$(l_r,\ldots,l_1)$	$(1,0)$	$(l_r,\ldots,l_1)$	$(1,r-1)$	β	$i=1, j=0$
$(l_r,\ldots,l_1)$	$(i,0)$	$(l_r,\ldots,l_1)$	$(1,r-1)$	$l_1\beta$	$1\leq i\leq r, j=0$
$(l_r,\ldots,l_i,\ldots,l_1)$	$(i,0)$	$(l_r,\ldots,l_i-1,l_{i-1}+1,\ldots,l_1+1)$	$(1,r-1)$	$i\alpha$	$1\leq i\leq r, j=0$
$(l_r,\ldots,l_b,\ldots,l_1)$	$(i,0)$	$(l_r,\ldots,l_b-1,l_{b-1}+1,\ldots,l_1+1)$	$(1,r-1)$	$bl_b\alpha$	$1<b\leq r$
$(l_r,\ldots,l_b,\ldots,l_1)$	(i,j)	$(l_r,\ldots,l_b-1,\ldots,l_{b-j}+1,\ldots,l_{i+j}+1,\ldots,l_i-1,\ldots,l_1)$	$(b-j,r-b+j)$	$bl_b\mu$	$1<b\leq r, j<b$

waiting in the queue, and $r-i-j$ replicas already failed. $Y(t)$ therefore takes value in the set $N_Y = \{(i,j)|1 \leq i+j \leq r, i \geq 1, j \geq 0\}$. To determine the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution, we follow the steps as described in the $r=2$ case in Section 4.1 to find the queue-length distribution $\boldsymbol{\pi}$, which then allows us to determine vector $\boldsymbol{\pi}(0)$, the distribution of the phase at the beginning of an all-busy period. We then solve Eq. (2.9) to find the matrix T that defines the matrix exponential representation $\boldsymbol{\pi}(x) = \boldsymbol{\pi}(0)\exp(Tx)$ of the stationary version of the process $\{X(t), J(t)\}$. Thus we define matrices S and $A^{(\text{jump})}$, which hold the transition rates of the service process associated to transitions without and with the start of a new job service, respectively, as summarized in Tables 4.5 and 4.6. For matrix S the first row considers the case where one of the l_1 jobs with a single replica in service either completes service or fails, with rate $\beta = \mu + \alpha$, allowing one of the j replicas of the youngest job in the queue to start service. In the second row, one of the l_b jobs with b replicas in service terminates successfully, canceling its siblings, and letting b new replicas to start service. Notice that the number of replicas of the youngest job waiting (j) must be at least b to ensure that no new job starts service. Similarly, if one of the b replicas of either of these l_b jobs fails, with rate α, one replica in the queue starts service. The last row for this matrix covers the case where one of the i replicas of the youngest job fails, allowing one of its siblings waiting to start service. For the $A^{(\text{jump})}$ matrix we consider similar scenarios, the main difference being that the number of replicas of the youngest job waiting j is assumed to be zero in most of the cases, as this implies that the next replica to join service will be part of a new job, and the transition thus corresponds to $A^{(\text{jump})}$. The only exception is in the last row, where a job with b replicas finishes successfully, and since $j < b$, this allows the j replicas of the youngest job in the queue, and $b-j$ replicas of a new job, to start service. After obtaining vector $\boldsymbol{\pi}(0)$ and matrix T, we follow the analysis of the $r=2$ case to obtain the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the



(a) Reliability: 90% (b) Reliability: 50% (c) Reliability: 10%

Figure 4.6: CCDFs of response times with the distributed set-up queues and with a centralized set-up (Poisson arrivals, $r = 2$, utilization:0.3)



(a) Reliability: 90% (b) Reliability: 50% (c) Reliability: 10%

Figure 4.7: CCDFs of response times with the distributed set-up queues and with a centralized set-up (Poisson arrivals, $r = 3$, utilization:0.3)

waiting-time process.

4.2.2 The service-time distribution

The next step is to show that the service-time distribution has a PH distribution with parameters $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$. For the service time, we focus on a tagged job in service, and let $Y(t)$ to be the service phase of this job at time t. Here $Y(t)$ takes values from $N_Y = \{(i,j)|1 \leq i+j \leq r, i \geq 1, j \geq 0\}$, as defined before. Clearly, the service time of a tagged job is affected by other jobs in service. Specifically, when the tagged job has replicas waiting in the queue, these replicas can only start service if one of the replicas in service frees a server. Thus it is essential to keep track of the states of all jobs in service, which can be done by means of $S(t) = (l_{r-1}(t), \ldots, l_1(t))$, where $l_i(t)$ is the number of jobs with i replicas in service at time t, *excluding the tagged job*.

Similarly to the case with individual queues, we define two absorbing states S and F that represent the cases where the job completes service successfully or encounters a failure, and describe the evolution of the service process as an MC with generator S_{ser}, given in Table 4.7.

We split the transitions in three sets, and the first case in any of these sets considers the successful completion of the tagged job caused by a successful tagged replica. The first set considers the case where the tagged job has zero replicas waiting in the queue. Thus, if any of its i replicas in service fails (second row) the number of replicas in service decreases by one. In case there is only one tagged replica in service, $i = 1$, a failure triggers the failure of the whole job (third row). The second set covers the special case where 1 tagged replica is waiting in the queue while the other $r - 1$ are in service. In the second row we consider the successful completion or failure, with rate $\beta = \mu + \alpha$, of the only non-tagged replica in service, which allows the tagged replica in the queue to start service. The third row instead considers the failure of one of the tagged replicas, which also allows the tagged replica in the queue to start service. Notice that in the first two sets the evolution is independent of the state of the other jobs in service, either because there are zero tagged replicas in the queue, or because there are $r - 1$ tagged replicas in execution. The other cases, which depend on the state of the non-tagged jobs in service, are described in the third set, starting with the successful completion of any of the i tagged replicas in service. In the second row, one of the l_1 jobs with a single replica in service either completes service successfully or fails, allowing one more tagged replica to start service. In the third and fourth rows, one of the l_b jobs with $b > 1$ replicas in service completes successfully, allowing b new replicas to start service. If there are at least as many tagged replicas as free servers, i.e. $j \geq b$, as in the third row, we are left with $j - b$ tagged replicas waiting in queue. If not, $j < b$ as in the fourth row, all tagged replicas waiting in the queue start service and we can just focus on the tagged replicas in service, ignoring any other jobs. In the last row, one of the b replicas of the l_b jobs in service fails, with $b > 1$, allowing a new tagged replica to start service.

Having obtained S_{ser}, we can obtain from the queue-length process the initial probability vector $\boldsymbol{s}_{\text{ser}}$ as in Eq. (4.2). This can be done following the analysis of the $r = 2$ case, by first finding matrices $Q_A(k)$ and $Q_S(k)$ that hold the rates at which a new job starts service due to an arrival or to a service completion in level k, respectively. We can then obtain the PH representation of the service-time distribution for successful jobs by directly applying Proposition 4, and finally, the PH representation of the response-time distribution $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$ by Theorem 3.1.

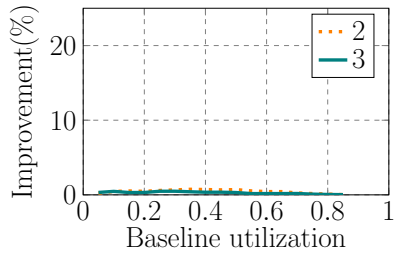
Table 4.7: Transition rates of S_{ser}

Condition	From		To		Rate	Condition
	$(l_{r-1}(t),\ldots,l_1(t))$	$Y(t)$	$(l_{r-1}(t),\ldots,l_1(t))$	$Y(t)$		
$j=0$	/	$(i,0)$	/	S	$i\mu$	$1\leq i\leq r$
	/	$(i,0)$	/	$(i-1,0)$	$i\alpha$	$1<i\leq r$
	/	$(1,0)$	/	F	α	$i=1$
$r-i=1$	/	$(r-1,1)$	/	S	$(r-1)\mu$	/
	/	$(r-1,1)$	/	$(r,0)$	β	/
	/	$(r-1,1)$	/	$(r-1,0)$	$(r-1)\alpha$	/
$r-i>1, j\geq 1$	$(l_{r-1},\ldots,l_i,\ldots,l_1)$	(i,j)	/	S	$i\mu$	$1\leq i\leq r$
	$(l_{r-1},\ldots,l_1)$	(i,j)	$(l_{r-1},\ldots,l_1-1)$	$(i+1,j-1)$	$l_1\beta$	$j\geq 1$
	$(l_{r-1},\ldots,l_b,\ldots,l_1)$	(i,j)	$(l_{r-1},\ldots,l_b-1,l_{b-1}+1,\ldots,l_1)$	$(i+b,j-b)$	$bl_b\mu$	$1<b\leq r, j\geq b$
	$(l_r,\ldots,l_b,\ldots,l_1)$	(i,j)	/	$(i+b,0)$	$bl_b\mu$	$1<b\leq r, j<b$
	$(l_r,\ldots,l_b,\ldots,l_1)$	(i,j)	$(l_r,\ldots,l_b-1,l_{b-1}+1,\ldots,l_1)$	$(i+1,j-1)$	$bl_b\alpha$	$1<b\leq r, 1\leq j<b$

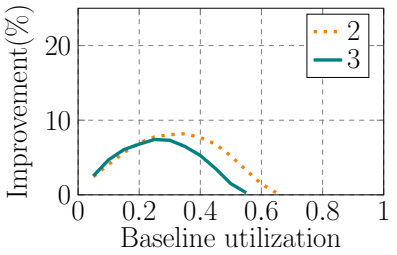
4.3 Comparison: distributed vs centralized

In this section, we compare the performance of the distributed set-up against the centralized set-up, and determine the performance gains obtained by keeping a central queue. Again, the service rate μ is set to 1, and the mean arrival and failure rates are set in proportion to μ to obtain different load levels and the baseline reliability.

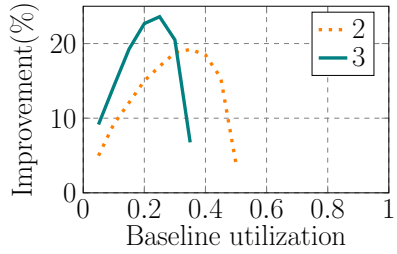
Given our independence assumptions, the distributed and centralized set-ups perform identically in terms of reliability, as this depends only on the replication level r. In terms of response times, the centralized set-up must achieve lower response times than its distributed counterpart, as in the latter case it is possible for some servers to have non-zero waiting lines, while other servers remain idle. This is not possible in the centralized set-up, as all servers must be busy if there is any job waiting to start service. To better understand this difference in performance we show in Figure 4.6(a) the complementary CDFs (CCDFs) of the response times achieved with 2 replicas by these two models, assuming Poisson arrivals, a baseline reliability of 90%, and a baseline load of 0.3 as an example. Clearly, it is hard to tell the difference between these two CCDFs. Reducing the reliability to 50% and 10%, Figure 4.6(b)-(c) shows a more significant improvement of the centralized over the distributed set-up along the whole distribution. Increasing the replication level to $r = 3$, Figure 4.7 shows that when the reliability is high, the distributed set-up performs almost as well as the centralized one, while this difference increases when the reliability decreases. Figures 4.6 and 4.7 also show that the improvement is not uniform along the distribution, and that it is more significant in the tail than in the body (e.g. second and third quartiles).



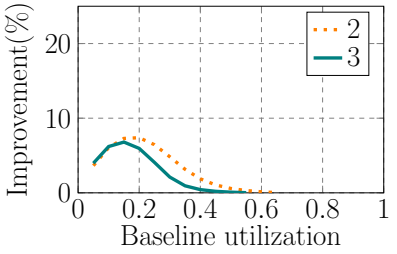
(a) Poisson arrivals, 90% reliability



(b) Poisson arrivals, 50% reliability



(c) Poisson arrivals, 10% reliability



(d) MAP arrivals, 50% reliability

Figure 4.8: Improvement on R_{95} of the centralized set-up over the distributed set-up

Considering different baseline utilization levels, Figure 4.8(a) examines an reliability of 90%, where the centralized set-up shows a relative improvement on the R_{95} of less than 1% over the distributed case. As discussed above, Figure 4.8(b)-(c) shows that reducing the reliability increases the advantage of the centralized set-up, and this can be over 20% for utilization around 0.2 and $r = 3$ when the reliability is just 10%. If we modify the arrival process considered in Figure 4.8(b) from a standard Poisson to a correlated MAP, Figure 4.8(d) shows that the gains obtained with the centralized operation are similar in magnitude but are restricted to a smaller subset of values for the baseline utilization, as with MAP arrivals replication is beneficial for a more limited load range. Similar results, both in trend and magnitude, can be observed if we compare the *mean* response time or other percentiles instead of the R_{95}.

From the experimental results we observed, as expected, that the centralized set-up achieves lower response times than the distributed case, especially at low baseline reliability. However, many large server farms dispatch incoming request to servers immediately, without holding a central queue. Replication with distributed queues is also required when accessing mirrored disks and searching distributed databases in parallel. This set-up provides more flexibility since there is no need to set up and manage a central dispatcher that keeps track of the servers state, making it easier to increase or decrease the number of servers as the load rises or falls.

A major point of our investigation is to assess the magnitude of the difference in response time between the two set-ups, to see if and when the distributed set-up suffers an excessive performance penalty. From our results, we observe that, although the centralized system performs better than the distributed set-up, this difference is quite small, under 1%, when the reliability is high (90%). Even if this reliability is 50%, which is already low as it assumes that one out of two requests fail, the improvement obtained with the centralized queue is at most 8%. We therefore observe that replication not only is effective to reduce the response times in the distributed set-up, as shown in Sections 3.3.4, but it also provides a performance close to that of the centralized set-up as long as the reliability is not too low. In addition, the proposed models focus on request failures, but not on server failures: the replicas in a job receive service from distinct servers, in the centralized set-up several replicas in a job may be submitted to the same server as the failure of one replica allows its sibling in the queue to start at the same server, increasing the risk of failure. The distributed set-up thus offers many benefits over the centralized case, while its performance disadvantage is limited when the reliability is high.

4.4 The case with parallel jobs

In the previous sections, we focus on systems with jobs that are composed of one task only. However, parallel job processing has become a common feature of many software systems. For instance, many scientific applications and data processing frameworks rely on splitting a job into many tasks, taking advantage of the large resource pools offered in grid and cloud infrastructures. When processing parallel jobs, the response time is determined by the last task to finish, a factor that can amplify the variability in the observed response times significantly, as observed in Google services [DB13]. This effect is further intensified when the tasks that compose a job fail, creating an overall unreliable job out of many highly-reliable tasks. To understand the impact of speculative replication has on the reliability and response time of parallel jobs, we propose a stochastic model to determine the response-time distribution in a system that processes synchronous parallel jobs and uses replication to improve reliability. The main features of the model are

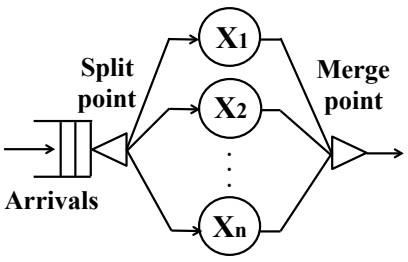


Figure 4.9: Split-merge queue

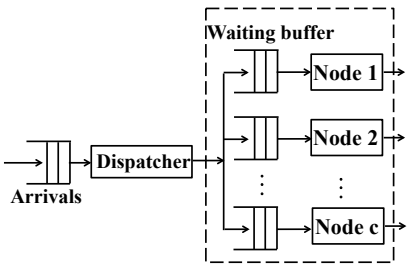


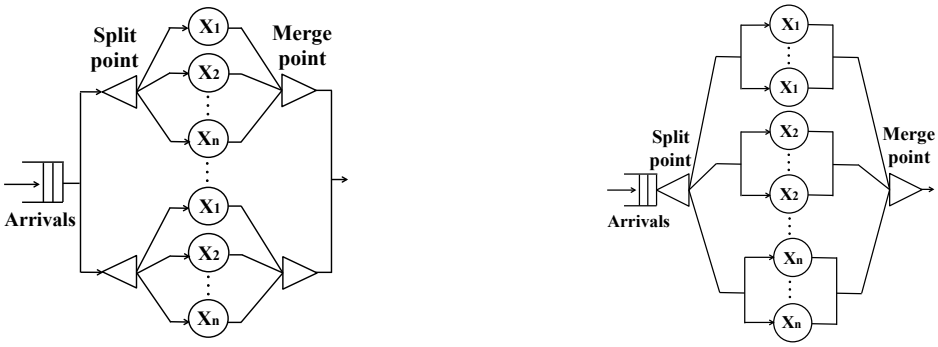
Figure 4.10: Reference Model

- It considers both task-level and job-level replication with canceling, as well as the no-replication case, in a single framework, as described in Section 4.4.2.
- It explicitly considers correlation between replica failures, covering cases from fully-independent to perfectly-correlated failures.
- It allows for general job arrival processes that can display different levels of variability and auto-correlation.
- In Section 4.4.4 we develop an efficient method to solve this model to obtain the job response-time distribution, which enables us to consider jobs with a large number of tasks.

Making use of this model, we have performed a number of experiments, reported in Section 4.4.7, to evaluate the impact of replication on the offered response times, and how this is affected by different system parameters.

4.4.1 Background

To model the execution of synchronous parallel jobs, and their replicas, we make use of split-merge queues [LK07]. In a split-merge queue, shown in Figure 4.9, an incoming job joins a queue and, upon reaching the head of the queue, it is split into multiple tasks, which are processed simultaneously by a set of parallel processors. These processors are all blocked until all the tasks finish service, at which point the tasks rejoin and depart from the system. Split-merge queues have been studied in different contexts. For example, [HZ07] analytically determined the job service time in a split-merge queue with random exponential service times, and introduced an approximation for general heterogeneous service times. Also, [LK07] derived an approximation for the response-time distribution in a split-merge queue with general service times and Poisson



(a) Job-level Replication (b) Task-level Replication

Figure 4.11: Reference job-level and task-level models

arrivals. These results however do not consider task failures nor replication, which are the focus of this section.

Reference model

In this section, we consider a system consisting of a central dispatcher and c distributed and homogeneous computing nodes, as shown in Figure 4.10, where jobs that arrive at the dispatcher are assigned to the computing nodes in either round-robin or random fashion. A job starts service directly if the corresponding node is idle, or joins the end of the queue in front of the node, if it is busy, and waits until all the jobs in front are processed, with FCFS scheduling. Each node is composed of multiple processors, such that, when a job starts service, each of its n tasks is assigned to one of the processors in the corresponding node. The number of tasks in a job, n, is also referred to as the *job size*. Similarly as previous sections, the task processing times are assumed to be exponentially distributed with rate μ, and tasks in service are subject to failures, with the time to failure being exponentially distributed with rate α. When a task fails during service, and no replication is adopted, the whole job fails, and leaves the computing node. To increase the system reliability, we consider two levels of replication granularity, as described next.

Replication granularity

For each arriving job, $r \geq 1$ replicas are submitted concurrently to the central dispatcher. In this section we consider the cases $r = 1$, where no replication is adopted, and $r = 2$,

where a total of two job replicas are submitted, including the original copy. We consider the two replication granularity levels proposed in [AGSS12] for parallel jobs. The first option is job-level replication, where for every job submitted to the system, two replicas are spawned and processed independently, as shown in Figure 4.11(a). Once the result from the earliest job-replica is obtained, the other one is canceled immediately if it is still in service. Job-level replication is appealing due to its simplicity, but each job-replica is successful only if all its tasks are successful. The second alternative is to clone at the task level, as shown in Figure 4.11(b), where every task is cloned and the result of the first replica to complete is returned, canceling its sibling if still in process. In this case it is enough that a single replica of each task succeeds to have a successful job completion.

In the case without replication, we assume the system is composed of c nodes and cn processors in total, such that it can process c jobs, each of size n, in parallel. If two replicas are adopted, then every two nodes, i.e., $2n$ processors, are grouped together, allowing the processing of $c/2$ concurrent replicated jobs.

4.4.2 The job service-time distribution

The service time of a parallel job is the interval between the time that all the job tasks enter a node's processors, and the time the job leaves the node, either with service completion or failure. In this section we show that, under the assumption of exponential service and failure times for the tasks, the *job* service time follows a PH distribution with representation $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$, either with or without replication. Further, we consider the case where failures across replicas are positively correlated, i.e., the failure of a replica makes more likely the failure of its partner.

No-replication model

In the no-replication case, the job service state is described by $N_1(t)$, the number of tasks in service at time t. The set of service states, or phases, is thus $N_J = \{n_1 | n_1 \in \{1, 2, \ldots, n\}\}$, and the job leaves the system when either all its tasks complete service successfully, or one of

Table 4.8: Transition rates for task-level replication

From	To	Rate	Range
(n_2, n_1)	(n_2-1, n_1+1)	$2n_2\alpha$	$n_2 \geq 1, n_1 \geq 0$
(n_2, n_1)	(n_2-1, n_1)	$2n_2\mu$	$n_2 \geq 2, n_1 \geq 0$
(n_2, n_1)	(n_2, n_1-1)	$n_1\mu$	$n_2 \geq 0, n_1 \geq 2$
(n_2, n_1)	(F)	$n_1\alpha$	$n_2 \geq 0, n_2 \geq 1$
(1,1)	(1,0)	μ	$n_2 = 1, n_1 = 1$
(1,1)	(0,1)	2μ	$n_2 = 1, n_1 = 1$
(1,0)	(S)	2μ	$n_2 = 1, n_1 = 0$
(0,1)	(S)	μ	$n_2 = 0, n_1 = 1$

them fails. The job service time thus has a PH($\boldsymbol{s}_{\text{ser}}, S_{\text{ser}}$) representation, where N_J is the set of transient states, and we consider two absorbing states, S and F, representing the cases where the job completes service successfully or encounters a failure, respectively. The absorption vectors $\boldsymbol{S}_S^*$ and $\boldsymbol{S}_F^*$ hold the absorption rates into states S and F, respectively, and $\boldsymbol{S}_{\text{ser}}^* = -S_{\text{ser}}\mathbf{1} = \boldsymbol{S}_S^* + \boldsymbol{S}_F^*$. If a job is in phase $n_1 > 1$, a task service completion leads the job to phase $n_1 - 1$ with rate $n_1\mu$. The (non-diagonal) entries of S_{ser} are thus $S_{\text{ser}}(n_1, n_1-1) = n_1\mu$. If $n_1 = 1$, the task service completion leads the job to a successful completion, thus to absorption in phase S. Instead, if the job is in phase $n_1 \in N_J$ and a task fails, the service process is absorbed in phase F with rate $n_1\alpha$. Finally, a job always starts service in phase n, as all its tasks start service at the same time, thus the initial probability vector is such that $\boldsymbol{s}_{\text{ser}}(n) = 1$, and all its other entries are zero.

Task-level replication

For task-level replication, we define the job service state to be $(N_2(t), N_1(t))$, where $N_2(t)$ is the number of tasks with both replicas in service, and $N_1(t)$ is the number of tasks with one failed replica at time t. As each job has n tasks, the set of service phases is $N_J = \{(n_2, n_1) | n_2, n_1 \in \{0, \ldots, n\}, 1 \leq n_2 + n_1 \leq n\}$, and the number of phases is $n(n+3)/2$. The transition rates of the sub-generator matrix S_{ser} are summarized in Table 4.8. Consider for instance the state $(n_2, n_1) \in N_J$ with $n_2 \geq 1$, where either of the two replicas of the n_2 tasks may encounter a failure, making the service process jump to state $(n_2 - 1, n_1 + 1)$ with rate $2n_2\alpha$. Further, since every task in the job starts with both replicas in process, the service process starts in state $(n, 0)$, thus $\boldsymbol{s}_{\text{ser}}((n, 0)) = 1$, and all the other entries of $\boldsymbol{s}_{\text{ser}}$ are zero.

Table 4.9: Transition rates for job-level replication

From	To	Rate	Range
(n_2, n_1)	$(n_2, n_1 - 1)$	$n_1\mu$	$n_2 > n_1 \geq 2$
(n_2, n_1)	$(n_2, n_1 - 1)$	$2n_1\mu$	$n_2 = n_1 \geq 2$
(n_2, n_1)	$(n_2 - 1, n_1)$	$n_2\mu$	$n_2 > n_1 \geq 1$
(n_2, n_1)	$(n_2, 0)$	$n_1\alpha$	$n_2 > n_1 \geq 1$
(n_2, n_1)	$(n_1, 0)$	$n_2\alpha$	$n_2 > n_1 \geq 1$
(n_2, n_1)	$(n_2, 0)$	$2n_1\alpha$	$n_2 = n_1 \geq 1$
$(n_2, 0)$	$(n_2 - 1, 0)$	$n_2\mu$	$n_2 \geq 2, n_1 = 0$
$(n_2, 0)$	(F)	$n_2\alpha$	$n_2 \geq 1, n_1 = 0$
$(n_2, 1)$	(S)	μ	$n_2 \geq 2, n_1 = 1$
(1,1)	(S)	2μ	$n_2 = 1, n_1 = 1$
(1,0)	(S)	μ	$n_2 = 1, n_1 = 0$

Job-level replication

For job-level replication, we track the number of tasks in service of each of the two job replicas as $(N_2(t), N_1(t))$. To limit the state space, we let $N_2(t)$ be the number of tasks in service of the job replica with *more* tasks in service, and let $N_1(t)$ be the number of tasks in service of the other job replica. The service-phase space is thus $N_J = \{(n_2, n_1) | n_2, n_1 \in \{0, \ldots, n\}, n_2 \geq n_1\}$, which has cardinality $n(n+3)/2$. In the example above, with $n = 2$, we have 5 service phases: {(2,2), (2,1), (2,0), (1,1), (1,0)}. Notice that when $N_1(t)$ equals 0, one of the job replicas has failed while the other is still in service. As an example, consider the case where the service process is in state $(n_2, n_1) \in N_J$ with $n_2 > n_1 \geq 1$, and one of the n_1 tasks of the job with the least number of tasks in process fails; the service process jumps to state $(n_2, 0)$ with rate $n_1\alpha$. This and the other transition rates for this case are summarized in Table 4.9. Also, as both job replicas start with all their tasks in process, all jobs start service in phase (n, n), thus $\boldsymbol{s}_{\text{ser}}((n, n)) = 1$.

Remark 1. *Notice that in all the cases, with or without replication, the S_{ser} matrix has an upper-triangular structure. In the case without replication we simply order the phase space in decreasing order, and see that the only transition allowed from state n_1 is to state n_1-1, resulting in an upper-triangular matrix S_{ser}. For the cases with replication, we obtain a similar structure by ordering the phase space lexicographically in decreasing order. From Tables 4.8 and 4.9 we observe that all transitions from a phase (n_2, n_1) are to a phase $(n_2, n_1 - 1)$ or to $(n_2 - 1, -)$, i.e., either the first or the second variable decrease. Thus, under a decreasing lexicographic order, the matrix S_{ser} is upper triangular. This structure will be exploited in the*

numerical methods proposed in Section 4.4.4.

4.4.3 Introducing correlation between replicas

So far we have assumed that the failures of the replicas of a task or job are statistically independent. However, this is not necessarily true for real applications, where the source of a failure may be located in the task itself, for instance due to its input data. We thus introduce correlation between replicas, which we model by defining the probability p that the failure of one replica causes its partner to fail as well.

Task-level replication

For task-level replication, the time to the first failure among the two task replicas is exponentially distributed with rate 2α. To incorporate correlation, we model the time to failure for the remaining replica as the product of two independent random variables, a Bernoulli random variable with mean $1-p$ and an exponential random variable with parameter α, i.e., the probability that the time to failure of the second replica is less than t is $p + (1-p)(1-\exp^{-\alpha t})$, for $0 \leq p \leq 1$ and $t \geq 0$. Clearly, the case when p equals 0 is equivalent to the independent case since the second replica fails with exponential rate α. Instead, when p equals 1, task-level replication works exactly as the no-replication case since the failure of either task replica leads to the failure of its sibling, resulting in the failure of the whole job. Intermediate cases are covered by values of p between 0 and 1, where a larger value of p decreases the job reliability as it makes more likely the failure of both task replicas.

Job-level replication

The introduction of correlation in job-level replication is more involved due to the lack of synchronization between the replicas, since the completion of a *task* replica does not cancel its partner. As the tasks are homogeneous, if a job is in service phase $(n_2, n_1) \in N_J$, for each of the n_2 tasks in the first job replica, its partner is still in service in the second job

Table 4.10: Transition rates for job-level replication with correlated failures

From	To	Rate	Range
(n_2, n_1)	$(n_2, n_1 - 1)$	$n_1\mu$	for $n_2 > n_1 \geq 2$
(n_2, n_1)	$(n_2, n_1 - 1)$	$2n_1\mu$	for $n_2 = n_1 \geq 2$
(n_2, n_1)	$(n_2 - 1, n_1)$	$n_2\mu$	for $n_2 > n_1 \geq 1$
(n_2, n_1)	$(n_2, 0)$	$n_1\alpha(1 - n_2p/n)$	for $n_2 > n_1 \geq 1$
(n_2, n_1)	$(n_1, 0)$	$n_2\alpha(1 - n_1p/n)$	for $n_2 > n_1 \geq 1$
(n_2, n_1)	(F)	$2n_1n_2\alpha p/n$	for $n_2 \geq n_1 \geq 1$
(n_2, n_1)	$(n_2, 0)$	$2n_1\alpha(1 - n_2p/n)$	for $n_2 = n_1 \geq 1$
$(n_2, 0)$	$(n_2 - 1, 0)$	$n_2\mu$	for $n_2 \geq 2, n_1 = 0$
$(n_2, 0)$	(F)	$n_2\alpha$	for $n_2 \geq 1, n_1 = 0$
$(n_2, 1)$	(S)	μ	for $n_2 \geq 2, n_1 = 1$
(1,1)	(S)	2μ	for $n_2 = 1, n_1 = 1$
(1,0)	(S)	μ	for $n_2 = 1, n_1 = 0$

replica with probability n_1/n, or it has already completed service with probability $(n - n_1)/n$. Thus, the failure of any of these n_2 tasks causes the failure of its sibling task in the other job replica with probability pn_1/n. As such an event triggers the failure of the whole job, the job jumps from phase (n_2, n_1) to the absorbing phase F with rate $n_2\alpha(pn_1/n)$. On the other hand, the failure of any of the n_2 tasks does not trigger the failure of its sibling in service with probability $(1 - p)n_1/n$. Also, the sibling task may have already completed service with probability $(n - n_1)/n$. As a result, the job transits from phase (n_2, n_1) to phase $(n_1, 0)$ with rate $n_2\alpha((1 - p)n_1/n + (n - n_1)/n) = n_2\alpha(1 - pn_1/n)$.

Using a similar argument, the failure of any of the n_1 tasks in the second job replica leads the job to transit from phase (n_2, n_1) to phase $(n_2, 0)$ with rate $n_1\alpha(1-pn_2/n)$, and to the absorbing state F with rate $n_1\alpha(pn_2/n)$. The transition rates for job-level replication with correlation are summarized in Table 4.10, from which it is clear that the case with $p = 0$ is equivalent to the independent case. However, when p equals 1, the behavior of job-level replication is not equivalent to the case without replication as the successful completion of a task in one job replica does not cancel its partner in the other job replica, which can still fail and make its job replica fail as well. We study this issue experimentally in Section 4.4.7.

Based on the analysis of a single node, the extension to the multi-node case relies on the simple observation that each node operates independently and only interacts with the other nodes via the arrival process, as described in Section 3. While we focus on the single-node case, the

analysis of the multi-node under round-robin scheduling case is obtained by substituting the arrival process MAP(m_a, D_0, D_1) by the MAP(cm_a, C_0, C_1), as in Eq. (3.1).

4.4.4 Efficient response-time computation

In Section 4.4.2 we obtained the PH representation $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ of the service-time distribution. Assuming we have a PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ for the waiting-time distribution, and given the independence between service and waiting times, we can obtain the PH representation $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$ of the response-time distribution following Eq. (2.21). This section therefore focuses on computing the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution, for which we propose a numerically-efficient algorithm that allows us to consider parallel jobs with a large number of tasks. In fact, this method is applicable for general MAP/APH/1 queues, where APH stands for acyclic PH distributions. APH distributions are characterized by having an upper-triangular generator matrix, which is a key feature of the job service time representations PH$(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ introduced in the previous section.

The waiting-time distribution

The waiting time of a job is the time period between its arrival and the time its service starts. To determine the waiting-time distribution, we follow the approach in [Sen89] as described in Section 2.5, observing the queue only during the busy periods, and defining a bivariate Markov process $\{(X(t), J(t))|t \geq 0\}$. Here the *age* $X(t)$ is the total time-in-system of the job in service, and the *phase* $J(t) = (A(t), N_2(t), N_1(t))$ holds the joint state of the arrival process $A(t)$ and the service process $(N_2(t), N_1(t))$, as defined in Section 4.4.2. The phase $J(t)$ thus takes values in a set of size $m = m_a m_s$. Recall that, for job-size n, in the no-replication case $m_s = n$, while under task- and job-level replication $m_s = n(n+3)/2$. Instead, the age $X(t)$ takes values in $[0, \infty)$, increasing linearly with rate 1 if no service completions occur. In case of a service completion, $X(t)$ suffers a downward jump, with its new value being equal to the waiting time of the job starting service. The service completion or failure of the *job* in service triggers the

job in front of the queue to start service, while the service completion or failure of a *task* does not necessarily trigger the start of a new job service.

To determine the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution, we rely on the stationary distribution $\boldsymbol{\pi}(x)$ of the $(X(t), J(t))$ process, which has a matrix-exponential representation as Eq. (2.8). Thus finding $\boldsymbol{\pi}(x)$ boils down to finding the $m \times m$ matrix T in Eq. (2.9) and the vector $\boldsymbol{\pi}(0)$. Recall that the off-diagonal entries of S hold the transition rates between the job service phases N_J, not associated with the start of a new job service, while $A^{(\text{jump})}$ holds the transition rates associated with a job service completion or failure, during a busy period. From the definition of S we see that it is in fact equal to S_{ser} since it holds transition rates that are not accompanied by a job service completion or failure. Similarly, $A^{(\text{jump})} = -(S_{\text{ser}}\mathbf{1})\boldsymbol{s}_{\text{ser}}$, as this matrix holds the rates with which a job completes service or fails, and a new one starts with phase according to $\boldsymbol{s}_{\text{ser}}$. Notice that, since $\boldsymbol{s}_{\text{ser}}$ has a single nonzero entry, $A^{(\text{jump})}$ has a single nonzero column. Also, as pointed out in Remark 1, the matrix S_{ser}, thus S, has an upper-triangular structure. We will exploit these features in developing an efficient algorithm to solve Eq. (2.9) to find T. After obtaining matrix T, we follow the steps in Section 2.5 to find the vector $\boldsymbol{\pi}(0)$, which is the stationary distribution of the phase at the beginning of a busy period, and then the PH representation $\text{PH}(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting time.

4.4.5 Exploiting restricted transitions

As the time complexity to solve Eq. (2.13) is cubic in m, and m increases quadratically with the number of tasks n, finding T becomes computationally expensive when considering instances with large job size n. We therefore introduce an efficient solution method to exploit the inner structure of the matrices S and $A^{(\text{jump})}$.

We consider a two-step approach. In the first step we exploit the structure in $A^{(\text{jump})} = -(S_{\text{ser}}\mathbf{1})\boldsymbol{s}_{\text{ser}}$, which has a single non-zero column since $\boldsymbol{s}_{\text{ser}}$ has a single non-zero entry. In the second step we exploit the upper-triangular structure in $S = S_{\text{ser}}$.

Exploiting the structure in $A^{(\mathbf{jump})}$

As the term $L(A^{(\text{jump})} \otimes D_1)$ in Eq. (2.11) has only $r = m_a$ non-zero columns, the iteration $T_{k+1} = S \otimes I_{m_a} + L_k(A^{(\text{jump})} \otimes D_1)$ only affects the first r columns of T, the last $m-r$ columns being given by the corresponding columns of $T_0 = S \otimes I_{m_a}$. To formalize this observation, we write $A^{(\text{jump})} \otimes I_{m_a} = \Gamma K$, where the $m \times r$ matrix Γ and the $r \times m$ matrix K are given by

$$\Gamma = -S_{\text{ser}}\mathbf{1} \otimes I_r, \text{ and } K = \begin{bmatrix} I_r & 0_{r \times m-r} \end{bmatrix}.$$

Letting $Y = L\Gamma$, we can rewrite Eq. (2.11) as

$$T = S \otimes I_{m_a} + L\Gamma K(I \otimes D_1) = S \otimes I_{m_a} + Y \begin{bmatrix} D_1 & 0_{r \times m-r} \end{bmatrix}.$$

Therefore, it is enough to compute the $m \times r$ matrix Y, instead of finding the $m \times m$ matrix L in each iteration. Focusing on the first r columns of this equation, we write $\tilde{T} = \boldsymbol{S}_1 \otimes I_r + YD_1$, where $\tilde{T}$ refers to the first r columns of the matrix T, and $\boldsymbol{S}_1$ is the first column of S. To find Y we solve

$$TY + YD_0 = -\Gamma, \tag{4.3}$$

which is obtained by post-multiplying Eq. (2.12) by Γ. Thus, using the Hessenberg-Schur method [GNVL79], we can solve Eq. (4.3) in $O(5m^3/3)$ time, where this leading term is due to the Hessenberg decomposition of T. Since solving the original Eq. (2.12) requires $O(115m^3/6)$ time, this step can be expected to provide a significant reduction in computation time, especially for large block-size m.

Exploiting the structure in S

To exploit the upper-triangular structure in S, we recall that the Hessenburg-Schur method to solve Eq. (4.3) requires finding the Schur decomposition of D_0, i.e., a quasi-upper-triangular matrix S and an orthogonal matrix V such that $D_0 = VSV'$. Thus, assuming this is the k^{th}

iteration, and post-multiplying Eq. (4.3) by V, we have

$$T_k Y V + Y V V' D_0 V = -\Gamma V.$$

Letting $X = YV$, $F = -\Gamma V$, and replacing T_k by its value from the previous iteration, we obtain

$$\left(S \otimes I_{m_a} + L_{k-1}(A^{(\text{jump})} \otimes D_1)\right) X + XS = F. \tag{4.4}$$

Following [GNVL79], the above equation can be solved by columns, by writing a linear system that involves finding one or two columns at a time, thanks to the quasi-upper-triangular nature of S. For the sake of clarity, we assume that S is upper-triangular, such that each step involves finding a single column. The more general case of a quasi-upper-triangular S can be treated similarly.

Using the upper-triangular structure in S we can find the p^{th} column of X, $\boldsymbol{X}_p$, by solving the system

$$\left(S \otimes I_{m_a} + L_{k-1}(A^{(\text{jump})} \otimes D_1) + S_{p,p} I_m\right) \boldsymbol{X}_p = \boldsymbol{F}_p, \tag{4.5}$$

where $\hat{\boldsymbol{F}}_p \equiv F_p - \sum_{i=1}^{p-1} \boldsymbol{X}_i S_{i,p}$. Notice that the right-hand side depends on the first $p-1$ columns of X, thus we can start with the first column and proceed forward finding one column at a time. This is a linear system of size m and becomes the bottleneck of the Hessenberg-Schur method, in which a Hessenberg decomposition of the matrix $(S \otimes I_{m_a} + L_{k-1}(A^{(\text{jump})} \otimes D_1))$ is used to find $\boldsymbol{X}_p$. Instead, we propose to take advantage of the structure in S. As this matrix is upper-triangular, the matrix $R = S \otimes I_{m_a} + I_m S_{p,p}$ is upper-triangular as well. Moreover, we know from the previous section that the matrix $L_{k-1}(A^{(\text{jump})} \otimes D_1)$ can be written as $L_{k-1}\Gamma K(I \otimes D_1) = Y_{k-1} D_1 K$. The left-hand side matrix in Eq. (4.5) can then be written as $R + Y_{k-1} D_1 K$, and since Y_{k-1} is an $m \times r$ matrix and $D_1 K$ is an $r \times m$ matrix, this expression is a rank-r correction of the upper-triangular matrix R. We can thus use the Woodbury matrix

identity [Hag89] to write the inverse of the matrix in Eq. (4.5) as

$$\left(R + L_{k-1}(A^{(\text{jump})} \otimes D_1)\right)^{-1} = R^{-1} - R^{-1}Y_{k-1}(I_r + D_1 K R^{-1} Y_{k-1})^{-1} D_1 K R^{-1}.$$

Further, post-multiplying by $\hat{\boldsymbol{F}}_p$ and recalling that $K = [I_r \ 0_{r\times m-r}]$, we can write

$$\boldsymbol{X}_p = (R + YK(I_{m_s} \otimes D_1))^{-1}\hat{\boldsymbol{F}}_p = R^{-1}\hat{\boldsymbol{F}}_p - R^{-1}Y_{k-1}(I_r + D_1\tilde{R}_Y)^{-1}D_1\tilde{R}_F,$$

where $\tilde{R}_Y$ holds the first r rows of the $m \times r$ matrix $R^{-1}Y_{k-1}$, and $\tilde{R}_F$ the first r rows of the vector $R^{-1}\hat{\boldsymbol{F}}_p$. As a result, the key step in finding the column $\boldsymbol{X}_p$ is the solution of the upper-triangular system $RZ = [\hat{\boldsymbol{F}}_p \ Y_{k-1}]$. With the Woodbury identity, this can be done in $O(rm^2 + r^2m + rm + r^3)$ time due to the triangular structure of R.

The solution of such system requires $O(rm^2)$ time, and this is done to find each of the r columns of X in Eq. (4.4), thus taking $O(r^2m^2)$ time to solve Eq. (4.4), which is equivalent to Eq. (2.12). This is a significant improvement compared to the standard method, which has a time complexity cubic in m, especially since r is n^2 times smaller than m, and n is the job size. The next section reports experiments that illustrate how significant the gains obtained with this method are.

4.4.6 Evaluation

To illustrate the efficiency of the proposed approach to compute the matrix T we compare it against the solution with the standard method described in Section 2.5.1, which we label **Full**. As the proposed method exploits the structure of both matrices $A^{(\text{jump})}$ and S, we present the results considering only the structure of matrix $A^{(\text{jump})}$, labeled **BS**, and considering both matrices, labeled **BW**.

The experiments were performed in MATLAB on an Intel Core i7-3770 machine, running at 3.4 GHz and with 16 GB of memory. We consider jobs with 50 and 100 tasks, service rate $\mu = 1$, and failure rate $\alpha = 0.1$. For the arrival process, we consider renewal processes with HE_2

Table 4.11: Computation times (sec) for jobs with 50 tasks

Arrivals (SCV)	U	**Full**	**BS**	**BW**	**Ratio**	
					R-BS	**R-BW**
HE_2 (SCV=2)	0.2	3583.6	91.8	1.7	39.0	2108.0
	0.5	7310.9	188.6	3.4	38.8	2150.3
	0.9	34634.0	878.0	15.8	39.4	2192.0
HE_2 (SCV=10)	0.2	4675.6	114.1	2.1	41.0	2226.5
	0.5	13041.3	325.5	6.0	40.1	2173.6
	0.9	/	915.2	15.9	/	/
MAP_2 (SCV=10) (decay = 0.1)	0.2	4613.5	121.3	2.0	38.0	2306.8
	0.5	13488.8	342.5	5.8	39.4	2325.7
	0.9	/	942.8	15.2	/	/
MAP_2 (SCV=10) (decay = 0.9)	0.2	5174.8	120.8	2.3	42.8	2249.9
	0.5	32208.6	742.8	13.2	43.36	2440.0
	0.9	/	3853.5	64.2	/	/

inter-arrival distributions and general order-2 MAPs. The parameters of the HE_2 distribution are computed using the moment-matching method in [Whi82], while the matrices D_0 and D_1 of the MAP are obtained with the method in [Hei04]. We consider MAPs with decay rate of the auto-correlation function equal to 0.1 and 0.9, and SCV equal to 10. With these parameters, the size of the phase space is $m = 2650$ for 50 tasks and $m = 10300$ for 100 tasks, while the number of nonzero columns in A_0 is $r = 2$ for both cases. We focus on task-level replication as the results are similar for the job-level replication case.

Table 4.11 summarizes the computation times to find the matrix T for the case of 50 tasks. The rows in Table 4.11 consider different arrival processes and utilization levels U, as shown in the first two columns. We set a time limit of 10 hours, and the results not shown here correspond to cases where the computation times exceeded this limit. We observe that the **BS** method reduces the computation times by two orders of magnitude compared with the **Full** method, while the **BW** method offers even larger reductions, of three orders of magnitude compared to the **Full** method. This can also be observed in the last two columns (Ratio), where we report the ratio between the **Full** and the **BS** and **BW** methods. Specifically, the computation times with the **BW** method are 2100-2500 times shorter than with the **Full** method, while the **BS** method offers a reduction of only 35-45 times.

Table 4.12 shows similar results for the 100-task case, but the results for the **Full** method are not reported since its computation times exceeded 10 hours in all instances. We show in the

Table 4.12: Computation times (sec) for jobs with 100 tasks

Arrivals (SCV)	U	**BS**	**BW**	**Ratio**
HE_2 (SCV=2)	0.2	5130.8	17.0	301.8
	0.5	8735.6	35.1	248.9
	0.9	/	156.8	/
HE_2 (SCV=10)	0.2	5219.7	20.3	257.1
	0.5	15237.1	56.3	334.2
	0.9	/	184.2	/
MAP_2 (SCV=10) (decay = 0.1)	0.2	5372.1	21.4	251.0
	0.5	18817.3	57.7	326.1
	0.9	/	177.9	/
MAP_2 (SCV=10) (decay = 0.9)	0.2	5696.3	22.4	254.3
	0.5	/	134.6	/
	0.9	/	595.2	/

last column the ratio of the computation times between the **BS** and the **BW** methods, where we observe that **BW** offers 240-340 shorter execution times than the **BS** method. Here and in the previous set of results we observe that the computation times of all the methods increase when the utilization, the arrival process variability and its auto-correlation increase. In the toughest case considered here, where the utilization is 0.9 and the arrival process SCV and decay rate of the auto-correlation function are 10 and 0.9, respectively, only the **BW** method is able to provide a result, and it does so in just under 10 minutes. In most of the other instances, its computation times are under 1 minute, highlighting its ability to efficiently compute T by exploiting the structure in the matrices $A^{(\text{jump})}$ and S.

Remark 2. *Notice that the solution method proposed here is not restricted to the analysis of parallel jobs considered in this section. For instance, using the results in [Cum82], any acyclic PH distribution (APH) can be transformed to have an entry vector $\boldsymbol{\tau}$ with all the mass in the first phase, and an upper-triangular matrix S. Thus, any MAP/APH/1 queue will fit within our assumptions regarding the structure of the S and $A^{(jump)}$ matrices.*

4.4.7 Evaluating the impact of replication

With the proposed model, and the efficient solution method introduced in the previous section, we are now in a position to evaluate the effect of replication on the reliability and response

times offered by the parallel-job computing cluster. In particular, we are able to determine how replication impacts these metrics under different system conditions, such as the system utilization, the correlation across replicas, and the job arrival process.

In the following we make use of the no-replication case as the baseline. The mean service rate μ is set to 1, while the failure rate α is set to achieve a target baseline reliability. Similarly, the arrival rate is set to achieve a target baseline utilization. To make the comparison fair, the baseline set-up has as many servers as the set-up with replication, which implies that, for job-size n, for every computing node with $2n$ servers in the scenarios with replication, there are two computing nodes in the baseline set-up, each with n servers.

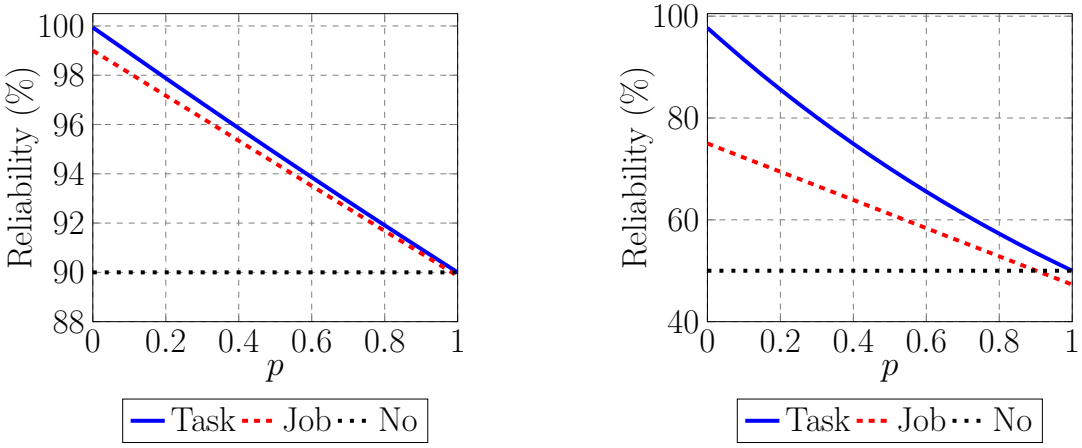
The impact of replication on reliability

The reliability is measured as the probability p_S that a job completes service successfully. When the replicas fail independently, the reliability depends only on the job size n, the service rate μ, and the failure rate α. In the no-replication case, it is given by $(\mu/(\mu+\alpha))^n$, while under task-level replication it is equal to $(1-(\alpha/(\alpha+\mu))^2)^n$, and for job-level replication it is $1-(1-(\mu/(\alpha+\mu))^n)^2$. Instead, when the failures between replicas are correlated, the reliability is

$$p_S = -\boldsymbol{s}_{\text{ser}} S_{\text{ser}}^{-1} \boldsymbol{S}_{\text{ser}}^*,$$

where $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ is the PH representation of the job service-time distribution, defined in Section 4.4.3, and $\boldsymbol{S}_{\text{ser}}^* = -S_{\text{ser}}\mathbf{1}$.

Figure 4.12(a) compares the job reliability under different replication modes, considering the case where the job-size n is 20, the baseline reliability is 90%, and the correlation p varies from 0 to 1. We observe that both task- and job-level replication improve the job-reliability under most values of p, with task-level having a slightly higher reliability. Also, the improvement in reliability decreases as the correlation p increases. Under task-level replication, the reliability decreases until it becomes equal to the case without replication when the correlation reaches 1. For instance, task-level replication improves the job reliability from 90% to 99.94% when



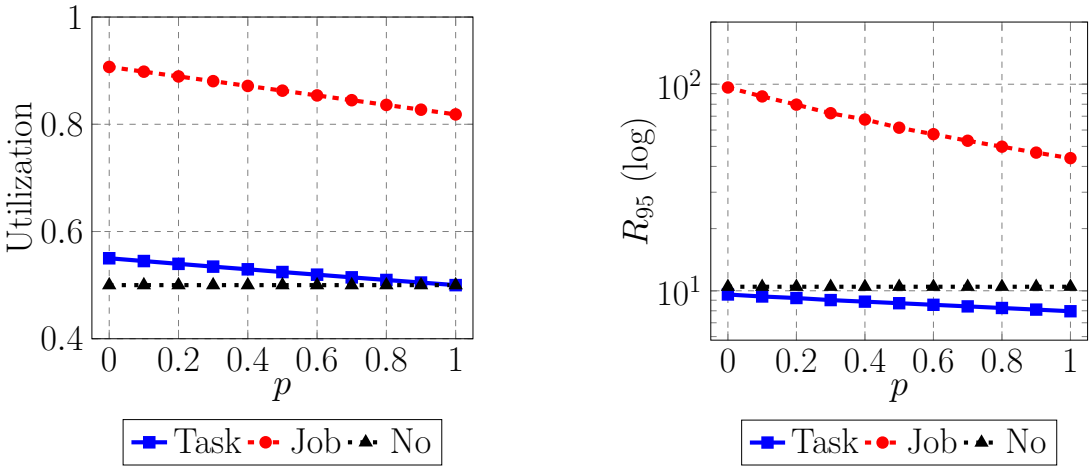
(a) Baseline reliability: 90%

(b) Baseline reliability: 50%

Figure 4.12: Reliability under different replication modes

$p = 0$, but decreases to 94.85% when $p = 0.5$, and drops back to 90% for $p = 1$. This effect is clearer in Figure 4.12(b), where the baseline reliability is decreased to 50%. Here the reliability achieved with task-level replication is significantly larger for small to medium correlations, and it is always better than no-replication, except under perfect correlation. However, we observe that the reliability achieved under job-level replication is much lower than the one achieved under task-level replication. For instance, when p is 0.1, the reliability achieved under task-level replication is 91.46%, while under job-level replication it is just 72.22%. This is because under job-level replication, a single task failure causes the failure of a whole job-replica, while in task-level replication the job fails only if *both* replicas of a task fail.

Figure 4.12 also offers a less expected, and somewhat counter-intuitive, result. While job-level replication clearly improves the reliability under a broad range of the correlation, it actually performs *worse* than the no-replication case for correlations close to one. The cause of this behavior lies in the lack of synchronization between the two job replicas. While in task-level replication the successful completion of a task immediately cancels its sibling, in job-level replication this synchronization does not occur. As a result, the sibling task continues in execution and may eventually fail, causing the failure of its job replica. This situation is exacerbated when the failure rate and correlation are high, degrading the job reliability until it becomes worse to replicate at the job level than to not replicate. This effect however only arises when the correlation is close to one, which is not to be expected in application-level failures.



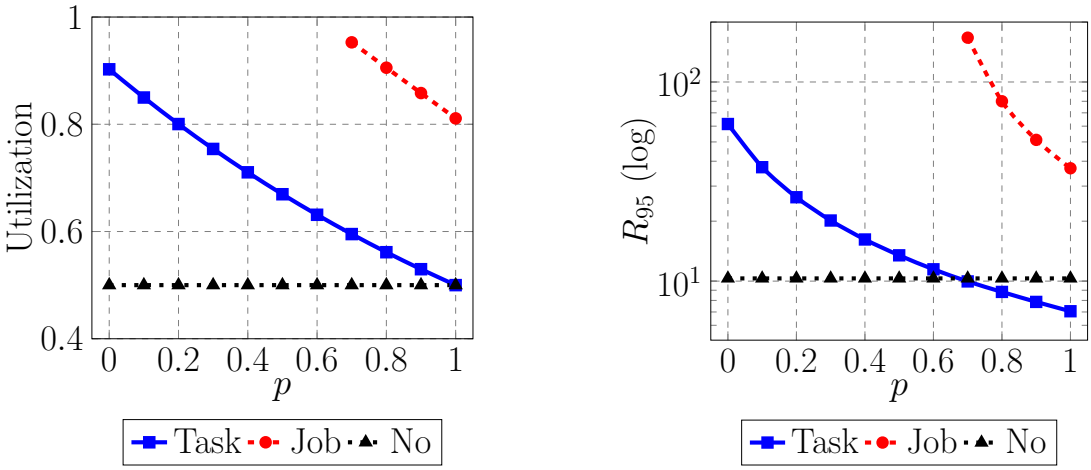
(a) Utilization (b) R_{95}

Figure 4.13: The effect of the correlation - baseline reliability: 90%

The impact of replication on the response times

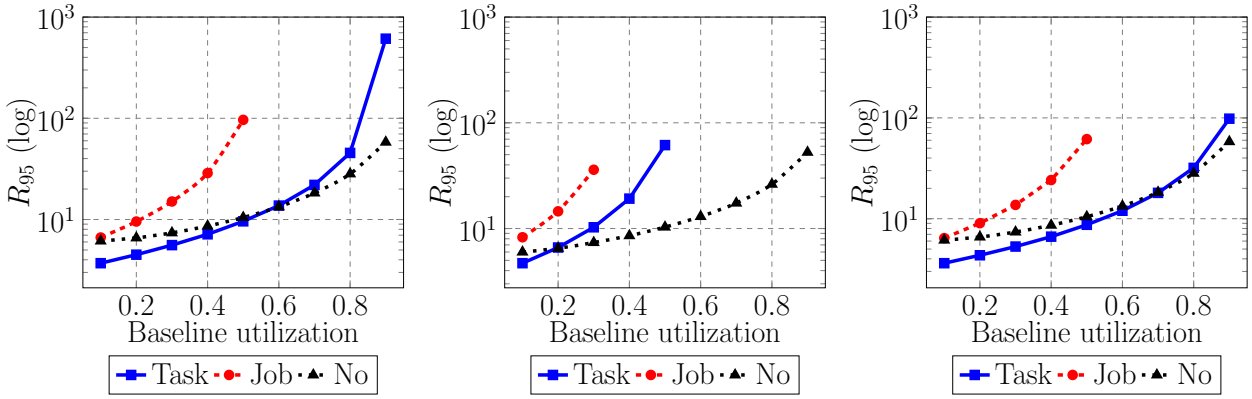
We now look further into the impact of replication on the offered response times, and how this effect depends on the system parameters. In particular, we focus on the response-time percentiles, where the i^{th} percentile of the response-time distribution, R_i, stands for the value x such that $i\%$ of the jobs experience a response time of at most x, for $i = 1, \ldots, 99$. We consider a case with job-size 20, HE_2 arrivals (SCV = 2), utilization of 0.5, and reliability of 90%. Figures 4.13(a) and 4.13(b) show how the utilization and R_{95}, respectively, change with an increasing failure correlation p. Clearly, both task- and job-level replication increase the system utilization, but job-level replication adds a much larger load, increasing the utilization from 0.5 to between 0.91 and 0.82. The effect of this additional load is clearly visible in Figure 4.13(b), which shows how the R_{95} shoots up under job-level replication, being one order of magnitude larger than under no-replication and task-level replication.

We now decrease the baseline reliability from 90% to 50%, and display the obtained utilization and R_{95} in Figures 4.14(a) and 4.14(b). In this case both task- and job-level replication increase the utilization compared to the no-replication case. In fact, when the correlation p is below 0.7, the additional load under job-level replication makes the system unstable. Recall that the utilization is 0.5 in this case. Moreover, Figure 4.14(b) shows that the R_{95} achieved under task-level replication is larger than without replication when the correlation p is below 0.7. This clearly differs from the previous case where task-level replication improved the R_{95} for



(a) Utilization (b) R_{95}

Figure 4.14: The effect of the correlation - baseline reliability: 50%



(a) Reliability: 90%; $p = 0$ (b) Reliability: 50%; $p = 0$ (c) Reliability: 90%; $p = 0.5$

Figure 4.15: R_{95} achieved with different baseline utilizations

every correlation level. This is because of two reasons:

- If the reliability is low and no replication is used, only short jobs can complete service successfully before encountering a failure. This causes the response times under no-replication to be necessarily short.
- Introducing replicas has two effects: first, it increases the resource utilization, thus increasing the response times; second, it allows the selection of the first replica that finishes, potentially reducing the response times. If the reliability is low, as in Figure 4.14, one of the replicas will likely fail and the first effect will be dominant, increasing the response times. Instead, if the reliability is high, as in Figure 4.13, then both replicas are likely to finish service, allowing the selection of the first result.

Moreover, we notice that both the utilization and the R_{95} decrease as the correlation p increases. Here again, as the correlation p increases it is more likely that the failure of a task leads to the failure of its replica, allowing only short jobs to complete service. Also, the larger correlation reduces the system load, allowing for shorter response times. In Figure 4.14(b) we also observe that, under large correlation values, even though the improvement in reliability is limited, task-level replication is still able to reduce the response times, thanks to its ability to select the first task-replica that completes.

The effect of the baseline utilization

As replication, even with canceling, increases the system load, it is natural to expect that its benefits can only be exploited as long as the baseline utilization is not too high. We now explore some scenarios to study how high can the utilization be for replication with canceling to effectively reduce the response times. Figure 4.15(a) compares the R_{95} achieved under different utilization levels, for the case with job-size 20, baseline reliability of 90%, HE_2 arrivals (SCV = 2), and uncorrelated failures ($p = 0$). Clearly, job-level replication achieves the highest R_{95}, increasing the response times compared to no-replication, and causing the system to become unstable when the utilization is larger than 0.5. Instead, task-level replication reduces the R_{95} compared to no-replication as long as the utilization is 0.6 or less. For larger values of the utilization, task-level replication increases the R_{95}, until the point that, under a 0.9 utilization, the R_{95} is actually one order of magnitude larger than without replication. It is still significant, though, that task-level replication causes just a small increase in the R_{95} for a utilization as high as 0.7, while offering a much better reliability (99.95%). If the baseline reliability decreases to 50%, as depicted in Figure 4.15(b), task-level replication is able to reduce the R_{95}, but only for a utilization of up to 0.2. For utilizations between 0.3 and 0.5, task-level replication offers a larger R_{95} than without replication, and higher values will make the system unstable. This is again the result of the low reliability.

We now go back to the case with reliability 90%, as in Figure 4.15(a), but introduce a failure correlation of $p = 0.5$, and depict the results in Figure 4.15(c). Comparing Figures 4.15(a)

and 4.15(c), we observe that task-level replication reduces the R_{95} up to a higher utilization (0.7) in the correlated case than in the uncorrelated case (0.6). This is because a larger correlation implies a higher probability that the failure of one replica causes its partner to fail, leading to a lower utilization under task-level replication, and therefore shorter response times. We thus observe that the impact of replication not only is affected by the utilization, but by the reliability and the correlation among the task failures. The results indicate that under low to medium utilizations, task-level replication is able to improve the response times, as long as the reliability is high. A low reliability scenario can benefit from replication in terms of reliability, but the room for improving the response times is more limited.

4.5 Discussion

Consistently high reliability and low latency are twin requirements common in server farms. In Chapters 3 and 4, we evaluated the effectiveness of the speculative replication strategy has on the system reliability and request latency in systems where requests are subject to failures during service. We developed stochastic models to determine the request response-time distribution for systems with both the distributed and centralized set-ups. We have observed from various experiments that replication has the potential to reduce the request latency, in addition to its benefit on the reliability, as long as the baseline utilization of the system stays below a certain threshold. This threshold, however, depends on a number of factors, including the arrival process, the request failure rate, the baseline utilization, and the number of replicas and servers. For example, the threshold diminishes under higher failure rates, and under bursty arrivals. Further, we observe that the improvement on latency is not uniform across different percentiles, and the load threshold under which replication helps to reduce latency largely depends on the metric targeted. Explicitly considering the response-time distribution, via the targeted percentiles, is therefore fundamental to evaluate a replication strategy.

Comparing the two set-ups, we find that the centralized set-up achieves lower response times than the distributed case. However, the distributed set-up offers response times close to those

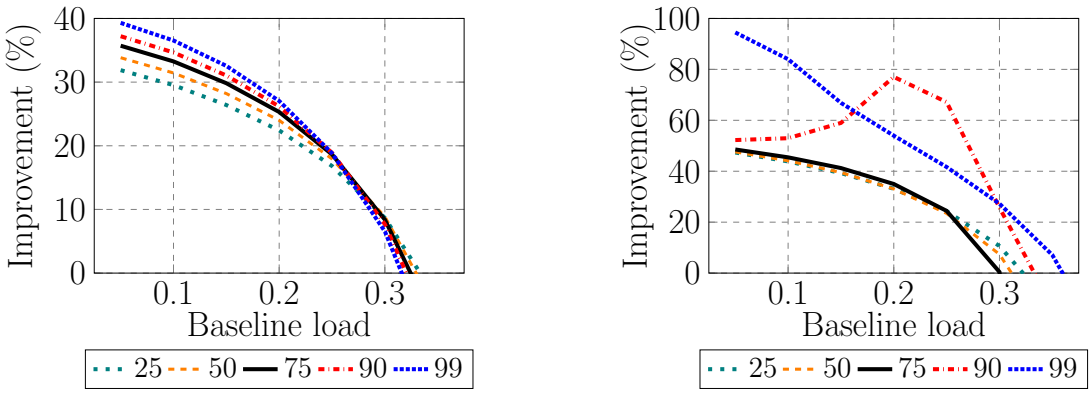
of the centralized system, especially at medium-to-high request reliability. In addition, the distributed set-up provides more flexibility since there is no need to set up and manage a central dispatcher that keeps track of the servers state, making it easier to increase or decrease the number of servers as the load rises or falls.

We then evaluated in Section 4.4 the ability of concurrent replication with canceling to improve the reliability and response times of parallel jobs subject to failures, where we demonstrate that task-level replication with canceling offers an effective way of improving the system reliability, with the potential to reduce latency, while limiting the increase in resource utilization.

Chapter 5

Variability-aware Replication for Latency-reduction in Reliable Systems

In the previous chapters, we have evaluated the effectiveness of the speculative replication strategy has on the system reliability and request latency in systems where requests are subject to failures during service. In this chapter, we focus on the impact of replication has on reducing the request latency, considering reliable systems, i.e., systems without failures. In addition, to be able to analyze a broad range of behaviors for the replica processing times, we will rely on Phase-type (PH) distributions, instead of considering exponential services only. In this chapter, in addition to evaluate the replication with canceling strategy, we are interested in analyzing replication for applications with very short processing times, such as web services, which process requests in the order of milliseconds. For *fast* systems, the canceling overhead would be comparable in magnitude to the replica processing time, such as web applications, in which requests usually take only milliseconds to respond. Therefore, canceling is hard or even infeasible to implement because of the non-negligible time required to cancel the replicas.



(a) SCV:0.5 (b) SCV:20

Figure 5.1: Improvement on percentiles

5.0.1 Motivating example

Replication offers a trade-off between the additional load introduced and the potential to reduce response time as it utilizes the first replica that completes service. Thus the main risk of replication is that it may negatively impact latency since it introduces extra load into the system, which may lead to excessive delays. In the previous sections, we have shown the effect of a number of system parameters, including the arrival process, the numbers of servers and replicas, and the failure rate, have on the effectiveness of replication. However, as the processing times are modeled as exponential, the effect of its variability is unclear. To illustrate this, let us consider a distributed system set-up with 50 servers, Poisson arrivals, and a short-tailed service process with low variability (measured by the SCV). In this low variability case, the SCV is set to 0.5. Figure 5.1(a) depicts the (positive) relative improvement (reduction) in the response times obtained by introducing replication without canceling ($r = 2$). We show the 25^{th}, 50^{th} and 75^{th} percentiles, also known as the three quartiles, and the 90^{th}, 99^{th} percentiles, representative of the tail, for an increasing baseline utilization. Clearly, replication provides significant gains under low utilizations, although the gain decreases with increasing utilization, and becomes negative when the utilization is larger than around 0.32.

Turning to service times with a long tail and high variability (SCV=20), Figure 5.1(b) shows that gains are much larger than in the low variability case, especially in the response time tail. Different from the low variability case, the gains across the percentiles are not uniform, and even the range of utilizations under which replication provides gains is different depending

on which metric is considered. For instance, under a 0.35 utilization, the 99^{th} percentile can be improved by close to 10% by introducing replication, but at this utilization replication is harmful to lower percentiles. As a result, if a decision is made to replicate based on possible gains on one percentile of the response-time distribution only, other percentiles may actually suffer. This highlights the importance of explicitly considering the variability of processing times, when evaluating a replication strategy. This is the purpose of the models introduced in the forthcoming sections.

However, evaluating the effect of replication for systems with MAP arrivals and PH processing times is non-trivial. In this chapter, we first focus on the *distributed* set-up in Section 5.1, where replication is achieved by running copies of requests on multiple servers that otherwise evolve independently, and waiting for the first replica to complete service. Here we consider the adoption of replication without canceling.

We then consider its counterpart – the centralized set-up, where we consider two cases in particular: the case where canceling is hard or almost infeasible to be implemented, and the case with canceling, in Sections 5.3 and 5.2, respectively. We show the effect that the statistical characteristics of service time, and in particular their *variability*, have on the potential for replication as a latency-tolerance mechanism. It is natural to expect that the service time variability, as noted for instance in [VGM+13, SLR16], should have an important role in the effectiveness of replication as a means of reducing latency. However, a number of questions remain unanswered when considering replication for latency-tolerance. In this chapter, we make use of the proposed models to answer the following important questions, by performing an in-depth analysis on how service times influence the effectiveness of request replication: for a given service time pattern (distribution), what is the maximum utilization at which replication can improve response times? Moreover, is such an improvement uniform over the response time distribution? For example, can we expect a decrease in the average response time as well as in its higher percentiles? While variability is a second-order effect, do higher order statistics, such as the service time's skewness, have a significant effect on the performance of replication? In addition, in which conditions is it possible to gain from more than one additional replica?

5.1 The distributed set-up: replication without canceling

In this section, we construct models that capture the evolution of a system adopting replication using approximate methods. In particular, we consider a distributed system where each server holds its own request queue and the role of the dispatcher is to assign requests and their replicas to these queues. In a reliable system, the synchronization introduced by the canceling mechanism allows a new job to start service with all its r tasks, as all r tasks execute simultaneously and the first one to complete cancels all other tasks. As a result, the system simplifies to a $MAP/PH/1$ queue with mean service rate $r\mu$ since we can aggregate the servers into a "super-server". We thus can rely on existing works as described in Section 2.5 to analyze this case with canceling. In this section, we focus on the more asynchronous and challenging case where canceling is hard or even infeasible to implement because of the non-negligible time required to cancel the replicas.

Analyzing this case is challenging as the servers work asynchronously but are not independent since for every new request several servers receive a copy at the same time. We therefore rely on approximate methods to cope with this set-up, which we use to analyze several scenarios quantitatively. Our main findings can be summarized as follows:

- The effect of replication is *non-uniform* across the response time percentiles and mean. In fact, there are cases where replication offers improvements in the mean but actually hurts the response time tail.
- Highly variable service times offer the best opportunity to reduce response times, especially in the tail. In this case it is even possible to obtain reductions in the high percentiles of the response time distribution (90^{th}, 95^{th}, 99^{th}) even if the mean response time worsens.
- An evaluation based on the exponential assumption for service times hides important effects of replication. The characteristics of response time under highly-variable service times, as typically observed in real traces, are in fact much more complex than under the exponential assumption.

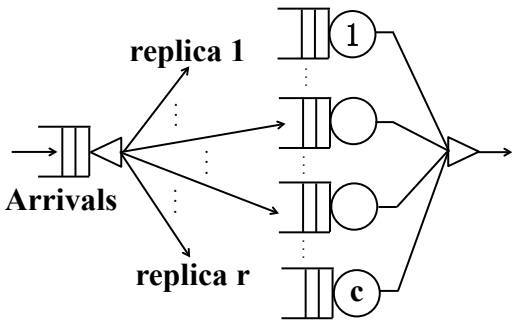


Figure 5.2: Replication model

- Beyond variability, the service time skewness also has a significant effect on the potential benefits of replication.

5.1.1 Problem set-up

We consider a system consisting of c parallel, homogeneous and distributed servers, each with an associated buffer, as shown in Figure 5.2. For each arriving request, a total of r copies are sent to r servers for processing. Replicas in front of each server form a single queue in the order of arrival and receive service with FCFS scheduling. The result of the request is returned by the first replica to complete service. To analyze this system, Section 5.1.2 starts by introducing a model to determine the request response-time distribution in the small-scale case where $c = r$, under fairly general inter-arrival and service times. This case can model, for instance, applications that maintain replicated data in a subset of resources, such that a request can be processed by any resource in this subset. Next, to analyze the general case where $c > r$, a different approach is proposed in Section 5.1.3, where we consider that the r replicas of each request are allocated to r out of c servers, chosen uniformly at random. This case, which we refer to as *random*, is more challenging to analyze and we therefore rely on the approximating assumption that replicas of a request are independent, to obtain the response-time distribution under fairly general service times. This case captures medium- or large-scale systems where servers are not dedicated to serve a single arrival flow.

Notice that the basic case where $c = r$ can be easily extended to consider the system with $c > r$ servers by assuming that the servers are partitioned into c/r groups of size r, with c a multiple of r. Thus, upon the arrival of a request, all its r replicas are assigned to one of the

groups, chosen at random. This allocation policy, which we refer to as *grouped*, can be found, for instance, in multi-core systems where a job is sent to one core utilizing all its threads and has the advantage of being easier to analyze as each group can be considered independently. However, we may expect this policy to offer worse performance than the random policy, as the latter benefits from greater multiplexing gains due to better load balance, with servers being idle less often.

In the reference model, we assume that requests arrive according to a MAP(m_a, D_0, D_1), while the replica service times follow a PH$(\boldsymbol{s}_{\text{rep}}, S_{\text{rep}})$ distribution. In the case of exponential service times, we assume a mean processing time $1/\mu$, such that $\boldsymbol{s}_{\text{rep}} = 1$, and $S_{\text{rep}} = -\mu$. The system utilization U is the expected fraction of the time that a server is busy, thus $U = r\lambda/\mu$, where λ is the mean arrival rate, and μ is the replica service rate.

5.1.2 Grouped allocation

To study the impact of replication under the grouped policy, we first introduce a stochastic model to determine the response-time *distribution* focusing on a single group with r servers, and later we show how the analysis can be extended to cover systems with multiple groups. The request response-time distribution is obtained by considering the service-time and waiting-time processes separately, and connecting them via a PH representation.

The waiting-time distribution

To determine the waiting-time distribution, we again perform an age-based analysis [AM01], observing the r queues only during the *all-busy* periods and define the bivariate Markov process $\{X(t), J(t)|t \geq 0\}$, with $X(t)$ the total time-in-system of the youngest *request* in service, and the joint phase of the arrival process $A(t)$ and the service state of the r queues $S(t)$, i.e., $J(t) = (A(t), S(t))$. To keep track of the state of the r queues, and taking advantage of their homogeneity, we focus on the queue-length differences with respect to the shortest one. The length of a queue includes the number of requests waiting and in service, where we truncate it

Table 5.1: Transition rates for S, $A^{(\text{jump})}$ and $S_{\text{not-all-busy}}$

Matrix	**From**	**To**	**Rate**
S	$(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_i, \ldots, \boldsymbol{n}_C)$	$(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_i - \boldsymbol{e}_j + \boldsymbol{e}_k, \ldots, \boldsymbol{n}_C)$	$n_{i,j} S_{\text{rep}}(j,k)$, $i \geq 0$
	$(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_{i-1}, \boldsymbol{n}_i, \ldots, \boldsymbol{n}_C)$	$(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_{i-1} + \boldsymbol{e}_k, \boldsymbol{n}_i - \boldsymbol{e}_j, \ldots, \boldsymbol{n}_C)$	$n_{i,j} S^*_{\text{rep}}(j) \boldsymbol{s}_{\text{rep}}(k)$, $i \geq 1$
$A^{(\text{jump})}$	$(\boldsymbol{n}_0, \boldsymbol{n}_1, \ldots, \boldsymbol{n}_{C-1}, \boldsymbol{n}_C)$	$(\boldsymbol{e}_k, \boldsymbol{n}_0 - \boldsymbol{e}_j, \boldsymbol{n}_1, \ldots, \boldsymbol{n}_{C-2}, \boldsymbol{n}_{C-1} + \boldsymbol{n}_C)$	$n_{0,j} S^*_{\text{rep}}(j) \boldsymbol{s}_{\text{rep}}(k)$
$S_{\text{not-all-busy}}$	$(n_0, \ldots, \boldsymbol{n}_i, \ldots, \boldsymbol{n}_C)$	$(n_0, \ldots, \boldsymbol{n}_i - \boldsymbol{e}_j + \boldsymbol{e}_k, \ldots, \boldsymbol{n}_C)$	$n_{i,j} S_{\text{rep}}(j,k)$, $i \geq 1$
	$(n_0, \ldots, \boldsymbol{n}_{i-1}, \boldsymbol{n}_i, \ldots, \boldsymbol{n}_C)$	$(n_0, \ldots, \boldsymbol{n}_{i-1} + \boldsymbol{e}_k, \boldsymbol{n}_i - \boldsymbol{e}_j, \ldots, \boldsymbol{n}_C)$	$n_{i,j} S^*_{\text{rep}}(j) \boldsymbol{s}_{\text{rep}}(k)$, $i \geq 2$
	$(n_0, \boldsymbol{n}_1 \ldots, \boldsymbol{n}_C)$	$(n_0 + 1, \boldsymbol{n}_1 - \boldsymbol{e}_j, \ldots, \boldsymbol{n}_C)$	$n_{1,j} S^*_{\text{rep}}(j)$

to be at most a constant C. Recall that the replica service-time distribution is $\text{PH}(\boldsymbol{s}_{\text{rep}}, S_{\text{rep}})$ with m_{rep} phases. Thus, we define $S(t) = (n_{i,j}(t), 0 \leq i \leq C, 1 \leq j \leq m_{\text{rep}})$, where $n_{i,j}(t)$ is the number of queues with queue-length difference i with respect to the shortest queue, and such that the replica in service is in phase j, at time t. Thus $S(t)$ takes values in the set $N_S = \{\boldsymbol{n} = (\boldsymbol{n}_1, \ldots, \boldsymbol{n}_C) | \boldsymbol{n}_i = (n_{i,1}, \ldots, n_{i,m_{\text{rep}}}), n_{i,j} \in \{0, \ldots, r\}, \sum_{i=0}^{C} \sum_{j=1}^{m_{\text{rep}}} n_{i,j} = r\}$, the cardinality of which is m_s. Section 5.1.2 evaluates the accuracy of this approximation and explores the value of C needed to ensure accurate results.

The next step is to determine the stationary distribution $\boldsymbol{\pi}(x)$ of the $(X(t), J(t))$ process, which has a matrix-exponential representation $\boldsymbol{\pi}(x) = \boldsymbol{\pi}(0) \exp(Tx)$, for $x > 0$. To determine the $m \times m$ matrix T, we need to find the matrices S and $A^{(\text{jump})}$, where the matrix $A^{(\text{jump})}$ holds the rates of service completion at any of the n_0 shortest queues, as a replica that completes service in these queues triggers the start of a new request service. Other transition rates are kept in the matrix S, as summarized in Table 5.1. The first row in this table considers the case where the replica in service goes through a service-phase transition from phase j to phase k, without completing service. The second row covers the case of service completions at any queue but the shortest one, where the replica in service phase j completes, and a new replica starts service in phase k. In the last row, a replica in any of the shortest queues, and service phase j, completes service, resulting in a single shortest queue, which is in phase k. Also, as the queue-length differences are bounded by C, transitions leading these differences to exceed the limit C are re-assigned to C.

To find $\boldsymbol{\pi}(0)$, which is the steady-state distribution of the phase at the beginning of an all-busy period, we need to model the system evolution during the not-all-busy period. During this period we only keep track of the arrival phase and the states of the queues, similar to the

phase in the all-busy-period, but in this case the shortest queues are empty, and therefore n_0 holds the number of idle servers. We thus define the process $\bar{S}(t)$, which takes values in the set $\bar{N}_S = \{(n_0, \boldsymbol{n}_1, \dots \boldsymbol{n}_C) | \boldsymbol{n}_i = (n_{i,1}, \dots, n_{i,m_{\text{rep}}}), 1 \leq i \leq C, n_0 \in \{1, \dots, r\}, n_{i,j} \in \{0, \dots, r\}, n_0 + \sum_{i=1}^{C} \sum_{j=1}^{m_{\text{rep}}} n_{i,j} = r\}$. In this case, the first entry of the vector $(n_0, \boldsymbol{n}_1, \dots \boldsymbol{n}_C)$ holds the number of idle servers, while the remaining entries are the same as for $S(t)$. The transition rates across service phases, between arrivals, during a not-all-busy period are held in the matrix $S_{\text{not-all-busy}}$, as summarized in Table 5.1. Here the first case corresponds to phase transitions without a service completion, while the second and third cases correspond to service completions.

Since $\boldsymbol{\pi}(0)$ is the distribution of the phase $J(t)$ at the beginning of an all-busy period, it solves

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0) \int_0^\infty \exp(Tu)(A^{(\text{jump})}_{\text{not-all-busy}} \otimes \exp(D_0 u)) du (S_{\text{not-all-busy}} \oplus D_0)^{-1} (K^{(start)} \otimes D_1). \quad (5.1)$$

The difference with respect to Eq. (3.13) lies in the matrix $A^{(\text{jump})}_{\text{not-all-busy}}$ and $K^{(start)}$, which connect the state space when going from an all-busy period to a not-all-busy period, and vice versa, respectively. The matrix $A^{(\text{jump})}_{\text{not-all-busy}}$ is similar to matrix $A^{(\text{jump})}$ in Table 5.1, but from a state $(\boldsymbol{n}_0, \boldsymbol{n}_1, \dots, \boldsymbol{n}_C) \in N_S$, the transition is to a state $(1, \boldsymbol{n}_1, \dots, \boldsymbol{n}_C) \in \bar{N}_S$ with rate $n_{0,j} S^*_{\text{rep}}(j)$. This transition corresponds to a service completion in the shortest queue, which starts a not-all-busy period with one empty queue. As the process enters a not-all-busy period, the phase evolves according to the sub-generator $S_{\text{not-all-busy}} \oplus D_0$, until an arrival occurs and triggers the process into a new all-busy period. When this occurs, the matrix $K^{(\text{start})}$ maps a state $(n_0, \boldsymbol{n}_1, \dots, \boldsymbol{n}_C) \in \bar{N}_S$ into a state $(\boldsymbol{n}_0, \boldsymbol{n}_1, \dots, \boldsymbol{n}_C) \in N_S$ with probability $p_{n_0, \boldsymbol{n}_0}$. As each of the n_0 idle servers selects its initial service phase according to $\boldsymbol{s}_{\text{rep}}$, the probability that the shortest queues start the all-busy period in state $\boldsymbol{n}_0 = (n_{0,1}, \dots n_{0,m_{\text{rep}}})$ follows a multinomial distribution, thus

$$p_{n_0, \boldsymbol{n}_0} = \frac{n_0!}{n_{0,1}! \cdots n_{0,m_{\text{rep}}}!} \boldsymbol{s}_{\text{rep}}(1)^{n_{0,1}} \cdots \boldsymbol{s}_{\text{rep}}(m_{\text{rep}})^{n_{0,m_{\text{rep}}}},$$

where $\boldsymbol{s}_{\text{rep}}(i)$ denotes the i^{th} entry of vector $\boldsymbol{s}_{\text{rep}}$. which is then used to find $\boldsymbol{\pi}(0)$ by solving a linear system.

After obtaining vector $\boldsymbol{\pi}(0)$ and matrix T, we are ready to find the waiting-time distribution, following the process described in Section 2.5. Notice here the expected number of arrivals in a not-all-busy period $E[\eta_1]$ is also 1, as an arrival during the not-all-busy period sends replicas to all queues in the group, initiating an all-busy period.

The service-time distribution

We now show that the request service times follow a PH distribution with parameters $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$. Let $N(t)$ be the service state of a tagged request in service at time t, which we define as $N(t) = (n_{i,j}(t), 0 \leq i \leq C, 1 \leq j \leq m_{\text{rep}})$, where $n_{0,j}(t)$ is the number of queues with a replica of the tagged request in service phase j, and $n_{i,j}(t)$, for $i > 0$, is the number of queues where the replica of the tagged request is in the i^{th} position in the queue. Notice that $N(t)$ is similar to the process $S(t)$, but it focuses on the queue-lengths *in front of the tagged request only*. Since the tagged request completes service when one of its replicas completes service, we are interested in the service phases in $N(t)$ where the r replicas of the tagged are still in service or waiting, thus $\sum_{i=0}^{C} \sum_{j=1}^{m_{\text{rep}}} n_{i,j} = r$. As a result, $N(t)$ takes values in the set N_S.

Next, to obtain the service time PH representation we describe the transitions among the states in N_S with a sub-generator S_{ser}. Notice that S_{ser} holds the service transition rates without a service completion in the shortest queue, thus it is equal to the matrix S in Table 5.1. The final step is to obtain the initial probability vector $\boldsymbol{s}_{\text{ser}}$, which is given by

$$\boldsymbol{s}_{\text{ser}} = [(1-\gamma)\boldsymbol{\pi}(0) + b\gamma\boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a})],$$

following Proposition 3, where $b^{-1} = \boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a})\mathbf{1}$ is a normalizing constant.

The response-time distribution

Based on the waiting and service-time distributions, we can obtain the response-time distribution in Theorem 5.1. The proof, which was done by Dr. Juan F. Pérez, can be found in Appendix A.1.

Theorem 5.1. *The job response time follows a PH distribution with parameters* $(\boldsymbol{s}_{res}, S_{res})$*, where*

$$\begin{aligned} \boldsymbol{s}_{res} &= \begin{bmatrix} (1-\gamma)\boldsymbol{\pi}(0) & \tilde{\boldsymbol{\alpha}}_{busy} & \mathbf{0}_{1\times m} \end{bmatrix}, \\ S_{res} &= \begin{bmatrix} S_{ser} & 0_{m\times m} & 0_{m\times m} \\ 0_{m\times m} & \tilde{S}_{ser} & (-\tilde{S}_{ser}\mathbf{1})P_{s,w} \\ 0_{m\times m} & 0_{m\times m} & S_{wait} \end{bmatrix}, \end{aligned} \tag{5.2}$$

where $\tilde{S}_{ser} = \Delta^{-1}S'_{ser}\Delta$*,* Δ *is a diagonal matrix such that* $\Delta\mathbf{1} = \boldsymbol{\delta}'$*, and* $\boldsymbol{\delta} = -\boldsymbol{s}_{ser}S_{ser}^{-1}$ *is the stationary distribution of the phase during service. Also,* $\tilde{\boldsymbol{\alpha}}_{busy} = (-S_{ser}\mathbf{1})'\Delta$*, and* $P_{s,w} = \Gamma^{-1}(T - S \otimes I_{m_a})'\Lambda$*, where* Γ *and* Λ *are diagonal matrices such that* $\Gamma\mathbf{1} = (T - S \otimes I_{m_a})'\Lambda\mathbf{1}$ *and* $\Lambda\mathbf{1} = \boldsymbol{\alpha}'_{busy}$*.*

The above analysis of a single group can be easily extended to the multi-group case, following Section 3.1.1. We refer to the random selection among groups as *grouped-random*, and we compare its performance against the random policy in Section 5.1.4.

Experimental validation

We evaluate the accuracy of the proposed model by comparing its results against simulation, focusing on the cases with $r = 2$ as an example, considering low (0.1), medium (0.5), and high (0.9) utilizations (achieved with replication), as well as different service and arrival processes. For the service process, we consider Exp, Erlang-2 (ER_2), and HE_2 distributions, thus covering a broad range of behaviors in terms of variability, measured by the SCV. The Exp and ER_2 distributions have SCV equal to 1 and 0.5, respectively, while for the HE_2 distribution we set it to 10. For the arrival process, we consider Poisson (Exp), and general order-2 MAP arrivals, where the SCV of the MAP inter-arrival times is set to 10 and the decay rate of the auto-correlation function is set to 0.5, thus modeling arrivals with high variability and auto-correlation. We compute the errors for different values of the limit C (100, 500 and 1000), and report the errors for some typical settings in Table 5.2. For each setting, the simulations were run for 5,000 times with 200,000 samples each time, from which we obtain the response-time mean and its 99^{th} percentile $R_2(99)$, and their 95% confidence intervals. Table 5.2 shows

Table 5.2: Approximation errors (Grouped)

Arr	**Ser**	**Utilization**	**Measure**	**Err(%)**		
				$C = 100$	$C = 500$	$C = 1000$
Exp	Exp	0.10	mean	<0.01	<0.01	<0.01
			$R_2(99)$	<0.01	<0.01	<0.01
		0.50	mean	<0.01	<0.01	<0.01
			$R_2(99)$	<0.01	<0.01	<0.01
		0.90	mean	<0.01	<0.01	<0.01
			$R_2(99)$	0.11	0.11	0.11
Exp	ER_2	0.10	mean	<0.01	<0.01	<0.01
			$R_2(99)$	<0.01	<0.01	<0.01
		0.50	mean	<0.01	<0.01	<0.01
			$R_2(99)$	<0.01	<0.01	<0.01
		0.90	mean	0.04	0.04	0.04
			$R_2(99)$	0.03	0.03	0.03
MAP	HE_2	0.10	mean	<0.01	<0.01	<0.01
			$R_2(99)$	0.05	0.05	<0.01
		0.50	mean	0.39	<0.01	<0.1
			$R_2(99)$	1.88	<0.05	<0.1
		0.90	mean	23.92	0.83	<0.1
			$R_2(99)$	26.06	0.14	<0.1

that, if C is large enough, the proposed model provides accurate results for both the mean and percentiles of the response times. Specifically, for the first two cases considered, Exp arrivals with Exp and ER_2 services, the errors are negligible with $C = 100$. This is due to the relatively short tails of these service-time distributions, which causes the queue-length differences to remain well within the $C = 100$ limit. The more challenging case with auto-correlated (MAP) arrivals and HE_2 services requires a larger limit C to achieve the same level of accuracy. The high variability in both the service and arrival processes, and the auto-correlated arrival stream, increase the probability of large queue-length differences. A larger utilization further increase these differences. Still, errors under 1% are achievable by setting $C = 500$ even under a 0.9 utilization.

5.1.3 An approximation for random allocation

We now consider the case where the request replicas are allocated to r out of the c servers, selected at random without replacement. Analyzing this system is challenging because it requires keeping track of the state of the c queues, as a request's replicas can be allocated to any subset of queues. Also, the evolution of these queues is correlated as every time a request

arrives, r queues receive one replica simultaneously. We thus opt for an approximation and show experimentally that it provides good results when the ratio c/r is large.

Approximation approach

To approximate the response-time distribution of this system we introduce the simplifying assumption that each queue evolves independently, receiving one request replica with probability $p = r/c$ every time a request arrives. This assumption was also used in [VGM+13] to approximate the *mean* latency in a system with a single replica. While this is a simplification of the actual system dynamics, we may expect this assumption to be reasonable when the number of servers c is sufficiently large compared to the number of replicas r. More specifically, if the request arrivals are modeled as a MAP(m_a, D_0, D_1), and the replica service times are PH, each queue can be modeled as an independent MAP/PH/1 queue with arrivals MAP(m_a, D_0', D_1'), where $D_0' = D_0 + (1-p)D_1$, $D_1' = pD_1$, and $p = r/c$.

Therefore, we can analyze each queue separately and obtain a PH representation $(\boldsymbol{\beta}_{\text{rep}}, B_{\text{rep}})$ of its response-time distribution following [Sen89]. Moreover, based on the independence assumption, the response time of a request can be obtained as the minimum of the independent response times of its r replicas. As the response time in each of the r servers is PH$(\boldsymbol{\beta}_{\text{rep}}, B_{\text{rep}})$, the request response time is PH$(\boldsymbol{\beta}_{\text{request}}, B_{\text{request}})$, given by the minimum of r independent PH variables as [LR99]

$$\boldsymbol{\beta}_{\text{request}} = \boldsymbol{\beta}_{\text{rep}} \otimes \cdots \otimes \boldsymbol{\beta}_{\text{rep}}, \quad B_{\text{request}} = B_{\text{rep}} \oplus \cdots \oplus B_{\text{rep}}.$$

Evaluating the approximation

We now evaluate the accuracy of this approximation by comparing its results against simulation, considering different utilization levels and c/r ratios, as well as the service-time distributions and arrival processes defined in Section 5.1.2. Table 5.3 shows the relative error in the mean and 99$^{\text{th}}$ percentile $R_r(99)$ of the response time for different settings. The first row considers

Table 5.3: Approximation errors (Random)

c/r	Arr	Ser	Utilization	Measure	r			
					2	3	4	5
2	Exp	ER_2	0.4	mean	3.14	4.91	5.87	6.54
				$R_r(99)$	7.73	12.74	15.82	17.93
25	Exp	ER_2	0.4	mean	0.27	0.48	0.39	0.38
				$R_r(99)$	0.61	1.09	1.15	1.19
25	Exp	ER_2	0.8	mean	0.54	0.88	1.32	1.37
				$R_r(99)$	0.97	1.35	2.85	2.90
25	Exp	Exp	0.8	mean	0.47	0.79	1.33	1.23
				$R_r(99)$	0.68	1.40	1.85	2.30
25	Exp	HE_2	0.8	mean	0.45	0.70	1.15	0.57
				$R_r(99)$	0.43	0.49	0.87	0.45
25	MAP	HE_2	0.8	mean	1.90	3.26	4.88	8.17
				$R_r(99)$	1.80	2.83	4.29	7.29
100	MAP	HE_2	0.8	mean	1.60	0.87	4.17	3.04
				$R_r(99)$	0.96	0.29	2.77	2.05
100	MAP	Exp	0.8	mean	1.84	2.98	4.39	5.84
				$R_r(99)$	2.54	4.99	7.43	9.91
100	MAP	ER_2	0.8	mean	1.66	4.10	5.09	6.08
				$R_r(99)$	3.20	7.70	9.73	12.18

a 0.4 utilization, Poisson arrivals and ER_2 service times, with a small ratio $c/r = 2$. We observe that the approximation produces errors lower than 8% for $r = 2$. However, the error increases with a larger r as the request response time depends on a larger number of correlated variables. Increasing the c/r ratio to 25, as shown in the second row, the errors decrease, with the maximum error being as low as 1.19% for $R_5(99)$ with ER_2 services. Increasing the utilization to 0.8 with $c/r = 25$, as shown in rows 3 to 5, for ER_2, Exp and HE_2 services, respectively, we observe that the errors increase with the utilization, although these remain below 3%. Clearly, for this and larger c/r ratios, the approximation provides high accuracy for both the mean and the tail of the response times. The next case in the table considers MAP arrivals and HE_2 services, which cause the errors to shoot up, compared to the case under Poisson arrivals. The reason is that MAP arrivals show high auto-correlation, while the approximation assumes independence among the response times of the replicas. To reach a higher accuracy, we increase the c/r ratio to 100 and the errors decrease to under 3% for HE_2 services. However, for Exp and especially ER_2 service times we observe larger errors. These results suggest that the approximation based on the independence assumption performs better with more variable service times as the variability decreases the dependence among the replicas' service times.

Threshold utilization & optimal number of replicas

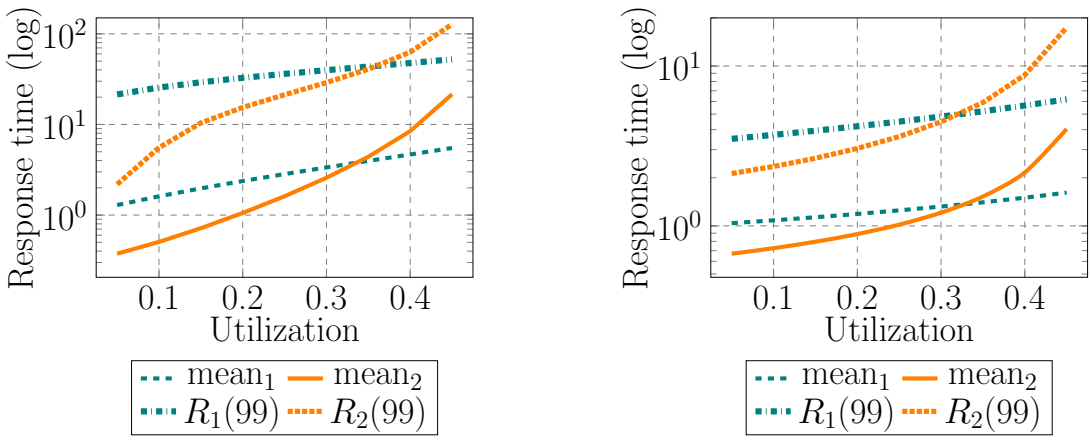
We now specialize the proposed approximation to the case of exponential services and Poisson arrivals, and try to answer two of the questions we posed in the introduction: Up to what utilization level can a reduction in response times be realized by implementing replication? What is the optimal number of replicas to adopt to minimize the response times? By focusing on exponential services and Poisson arrivals we can exploit closed-form expressions for the mean and the percentiles of the response times. As we shall see, however, the exponential assumption leads to an over-simplification of the effects of replication, and a better representation of the service times is necessary to fully assess its potential benefits.

Threshold utilization: Assuming r replicas are adopted for each request, we want to find the *threshold utilization*, which is the maximum baseline utilization under which replication is beneficial. The response-time distribution of a request with r replicas is $F_r(t) = 1 - \exp^{-r(\mu-\lambda^*)t}$, where $\lambda^* = r\lambda/c$ is the arrival rate to each queue, and the system arrival rate is λ. Thus the response time p^{th} percentile is $R_r(p) = \ln(1/(1-p))/(r\mu - r^2\lambda/c)$, where ln is the natural logarithm. If the adoption of replication results in a lower latency, then we have $R_r(p) \leq R_1(p)$, from which we obtain the threshold utilization as

$$\rho^* = 1/(r+1).$$

For example, when r=2 the threshold utilization is $\rho^* = 1/3$, i.e., only if the system utilization is below 1/3, the introduction of 2 replicas reduces the latency. Notice that the same threshold is found for the *mean* response times, as in [VGM+13] for $r = 2$. As a result, under exponential services and Poisson arrivals, the threshold utilization is independent of the specific response-time percentiles considered, and gains in the mean translates in gains in all the percentiles. As we show in Section 5.1.4, this conclusion is not valid for more general service times.

Optimal number of replicas: The p^{th} percentile $R_r(p)$ of the response-time distribution, can be shown to have a positive second derivative whenever the system is stable, i.e, $\mu > r\lambda/c$. Thus $R_r(p)$ is a convex function in r, and the optimal number of replicas to adopt is the



(a) HE_2 services (b) ER_2 services

Figure 5.3: Response time mean and percentiles under $r = 1, 2$

smallest number r^* beyond which the latency increases, i.e., r^* is the smallest r that satisfies $R_r(p) \leq R_{r+1}(p)$, thus

$$r^* = \min\left(\lfloor c\mu/\lambda \rfloor, \lfloor (c\mu - \lambda)/(2\lambda) \rfloor\right), \tag{5.3}$$

where the first condition ensures stability. In fact, we find that the same result holds for the *mean* response time. Thus, under exponential services and arrivals, the number of replicas can be chosen to be optimal for both the mean and every percentile of the response-time distribution. We will test this result for more general arrival processes and service times in the next section. To demonstrate this result, consider a system with 50 servers, mean arrival and service rates of 10 and 1, respectively. From Eq. (5.3) the optimal number of replicas to adopt is 2. Simulation results show that the mean response time when introducing 1, 2, 3 and 4 replicas is 1.251, 0.836, 0.839 and 1.275, respectively, while the 99^{th} percentile is 5.762, 3.857, 3.889 and 5.957, respectively. These results verify that $r = 2$ is the optimal number of replicas to adopt.

5.1.4 Experimental results

In this section we make use of the proposed models to evaluate the impact of replication on the offered response times. As we are particularly interested in the effect of service time variability, we make use of the ER_2, Exp, and HE_2 distributions introduced in Section 5.1.2. To look into the effect of the arrival process we consider both Exp IATs, and correlated MAP arrival

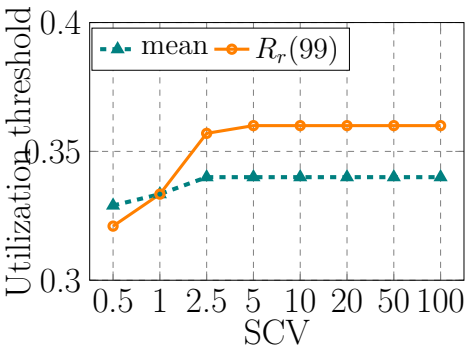


Figure 5.4: Utilization threshold (random)

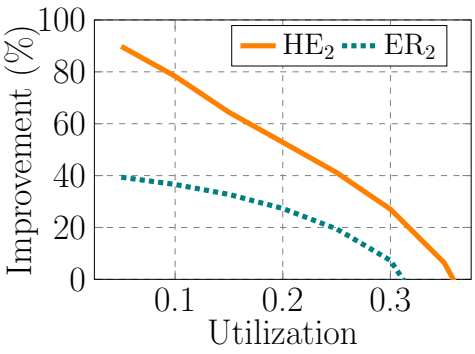


Figure 5.5: $R_r(99)$ improvement (random)

processes. Throughout, we assume $c = 50$ servers, and set the service rate μ to 1, while the arrival rate is set to achieve different utilization levels.

The utilization threshold

We start by investigating the utilization levels under which replication reduces latency, and whether this depends on the service time variability and the replica-allocation policy. Figure 5.3 shows the response time mean (mean_r) and 99$^{\text{th}}$ percentile ($\text{R}_r(99)$) obtained for $r = 1$ and 2, respectively, for baseline utilizations between 0.05 and 0.45, with random allocation. Notice that the utilization achieved after the introduction of replication doubles when $r = 2$. While Figure 5.3(a) focuses on highly variable HE_2 services, Figure 5.3(b) considers low-variability ER_2 service times. In both cases, replication clearly reduces both the mean and the 99$^{\text{th}}$ percentile of the response times, although the reduction decreases with increasing utilization, up to a utilization level where replication becomes harmful. Comparing these figures we observe that the utilization threshold is higher under HE_2 services than under ER_2. This is more explicit in Figure 5.4, which reports the utilization threshold for service times with an SCV that ranges from 0.5 to 100. Here we observe not only that the threshold increases with a larger variability, and stabilizes for SCVs larger than 5, but the threshold is actually different for the mean and the 99$^{\text{th}}$ percentile. As a result, for service times with an SCV greater than one, there is a region of the utilization where replication does not improve the mean response time but it still improves the 99$^{\text{th}}$ percentile.

In addition, Figure 5.3 also shows that the reduction in response times for HE_2 services is

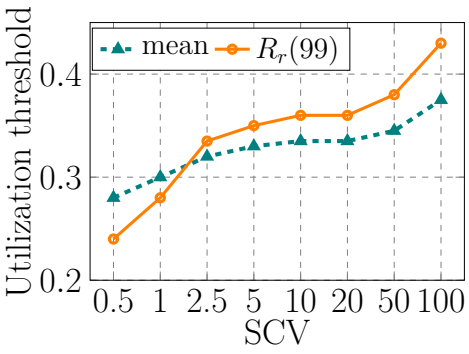


Figure 5.6: Utilization threshold (grouped)

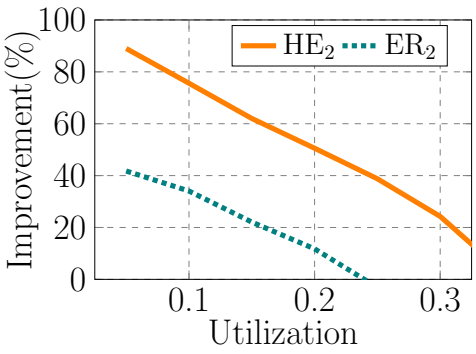


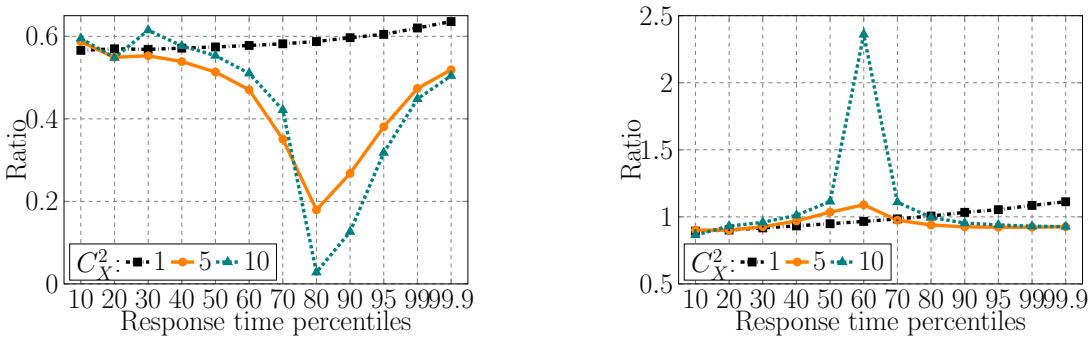
Figure 5.7: $R_r(99)$ improvement (grouped)

much larger than for ER_2 services. This is further highlighted in Figure 5.5, where we depict the relative improvement (reduction) in the response times 99$^{\text{th}}$ percentile when introducing replication. While under HE_2 services the gains can reach up to 90% under very low utilizations, under ER_2 services these are limited to at most 50%. This difference arises from the more concentrated service times in the case with lower variability, which offers fewer chances to have at least one replica with a short response time.

Similarly, under grouped-random allocation the threshold utilization is also higher for HE_2 than for ER_2 services, but the differences are much larger than under the random policy. In fact, if we evaluate the threshold utilization under service times with different variability, as in Figure 5.6, we observe that this threshold can be between 20% and 45%, a much wider span than under the random policy (see Figure 5.4). This figure also highlights that for lowly-variable service times (SCV 1 or below), a decision to replicate based on a potential gain of the mean response time can actually worsen the 99$^{\text{th}}$ percentile. Further, the improvements under highly variable (HE_2) service times are also more significant than under ER_2 services, as shown in Figure 5.7. The latter case provides gains of at most 40%, while the gains under HE_2 services can be as large as 90% and sustain to around 20% for utilizations as high as 0.3.

The effect of higher moments

We now look further into the impact of the service time variability on the effect of replication across the whole response time distribution. To this end we consider three values for the SCV ($C_X^2 = 1$, 5 and 10) and fix the first moment (mean) of the service time distribution



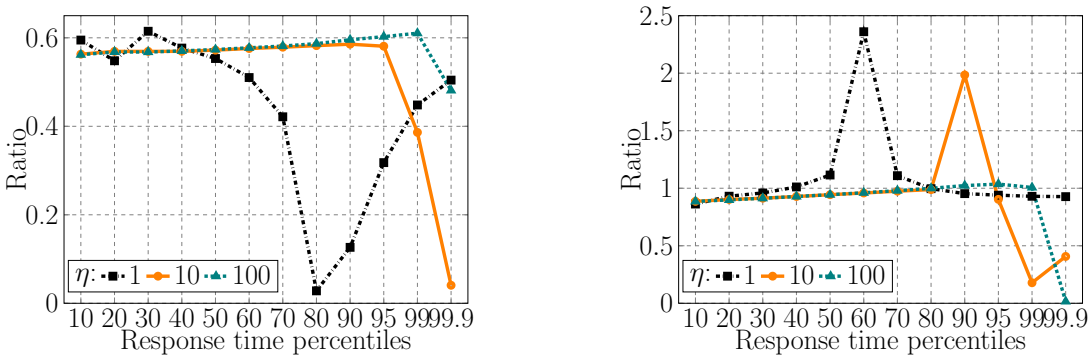
(a) Baseline utilization: 0.1

(b) Baseline utilization: 0.3

Figure 5.8: The impact of Variability

M_1 to match the utilization. The second and third moments of the distribution are obtained as $M_2 = (C_X^2 + 1)M_1^2$ and $M_3 = 3M_1^3(1 + C_X^2)^2/2$. We rely on the method in [BHT05] to match these first three moments with a PH distribution. For each percentile p in the range $\{10, 20, \ldots, 90, 95, 99, 99.9\}$ we obtain the ratio $R_2(p)/R_1(p)$, comparing the response times obtained with replication ($r = 2$) against those without ($r = 1$), with random allocation. Figure 5.8(a) depicts these ratios for a 0.1 utilization. The exponential case ($C_X^2 = 1$) shows a very smooth behavior, with ratios of about 60% across the whole percentile range. The more variable service times, however, show a strikingly different behavior, with ratios that vary wildly among the percentiles considered. For instance, for the 80^{th} percentile the ratio can be under 0.05 for the $C_X^2 = 10$ case, i.e., this percentile with replication is 20 times lower than without replication. The opposite case can be observed in Figure 5.8(b), where the utilization is 0.3. Here the exponential case is again very smooth, with a ratio of around 1 across the percentiles considered. However, with a larger service time variability we observe large peaks in the central percentiles. In this case replication can be beneficial for the response time tail, but it also increases the response times experienced by a significant proportion of requests.

Beyond the variability, we also consider the effect of the skewness, or third moment, which measures the asymmetry of the service time distribution. To this end we fix the first moment to match a given utilization, and the SCV to $C_X^2 = 10$, and define a baseline skewness coefficient as $\bar{\gamma} = 3C_X^2/2 + 1/(2C_X^3)$, which is the minimum skewness coefficient representable by a PH distribution of order 2. We then let the skewness coefficient be $\gamma = \bar{\gamma}\eta$, and consider values of η in the set $\{1, 10, 100\}$. Figure 5.9 shows the ratio $R_2(p)/R_1(p)$ for percentiles across the



(a) Baseline utilization: 0.1

(b) Baseline utilization: 0.3

Figure 5.9: The impact of Skewness

Table 5.4: Fitting error

Dis	Err%
Exp	5.004
HE_2	2.701
HE_3	0.812
HE_4	0.080
HE_5	0.049
HE_6	0.048

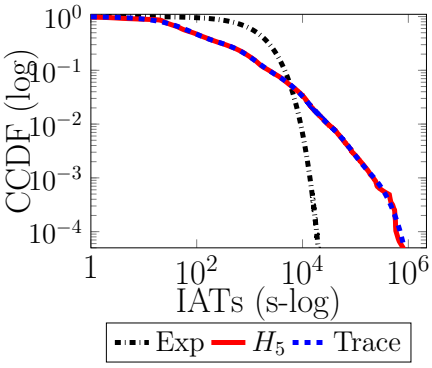


Figure 5.10: Log C services

response time distribution for the values of η given above. Of the three cases considered, the one closest to an exponential is $\eta = 100$, as most of the mass of the initial probability vector is concentrated in one of the phases. In this case the percentile ratio is very stable across the distribution. However, as the skewness decreases we observe very significant differences in the percentile ratio. In both utilization scenarios a smaller skewness implies that the main effect of replication concentrates on lower percentiles. For instance, with utilization 0.1 and $\eta = 10$ or 100, the main effect is on the tail, while for $\eta = 1$ the main effect is around percentile 80. Thus, in addition to the mean service time, higher moments need to be considered when implementing replication as they have a very significant effect on the offered response times.

Comparing allocation policies

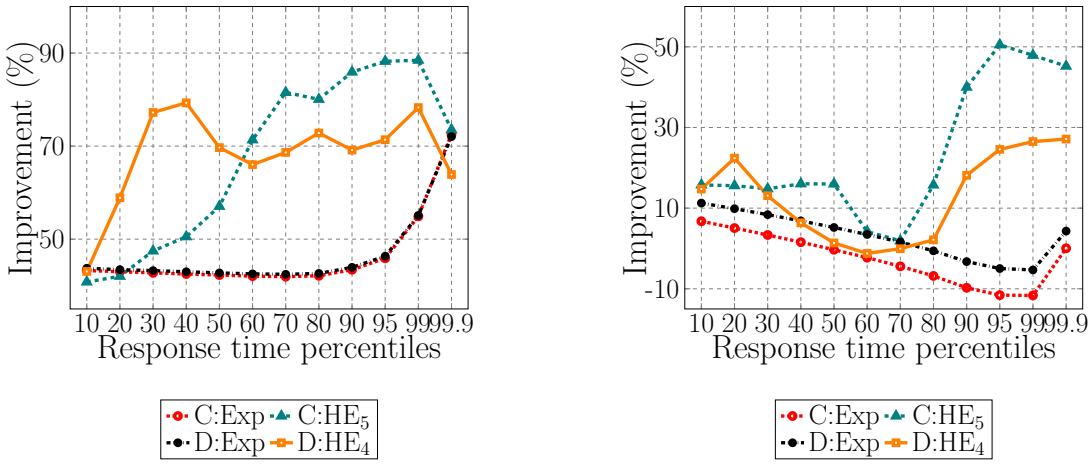
We also compare the performance of the two policies considered, random and grouped-random. The results, repeated over a wide range of scenarios, show that the grouped-random policy

performs worse than the random policy. This is caused by the greater multiplexing gains offered by the random policy, as it allocates the replicas of new requests to a completely random subset of servers, while in the grouped-random case the random selection is done at the level of server groups, each of which receives all the replicas for a given request.

Case study

We now consider realistic traffic patterns by making use of the Intel-NetBatch C and D logs [Par]. These logs contains records from the Intel Netbatch grid of one month, which is composed of multiple clusters in different locations around the world, each with tens of thousands of nodes. Each log contains the workload from one cluster, including job ID, job submit time, start time, end time, exit status and the number of cores used (number of tasks). We focus on successful single-task requests that amount to over 99% of all successful requests. We observe heavy-tailed service times in both logs, with SCV of 76.07 and 12.20, respectively. Given the high variability, we fit the service times using hyper-exponential distributions with r phases (HE_r), relying on the maximum-likelihood method in [KSH03], as implemented in jPhase [PR06]. Table 5.4 shows the mean relative error between the empirical CDF and the fitted CDFs for log C, which clearly decreases as the number of phases increases. For instance, the error is 5.00% for Exp (i.e., HE_1), and 0.05% for HE_5. The improvement is however limited after five phases, thus we choose the HE_5 representation for log C. Figure 5.10 shows the empirical CCDF of the service times, and the fitted exponential and HE distributions. Clearly, the HE distribution captures the behavior of the real traces much better than the exponential distribution, both the body and tail. Similarly, we fit the service times of log D using an HE_4 distribution. For the arrival process, we observe that the SCV is 1.92 and the decay rate of the auto-correlation function (ACF) is 0.97 for log C, which we capture with a MAP using the method in [Hei04]. For log D, the SCV is 8.43 and the ACF is just 0.09, thus we use an HE distribution to match the high variability and the low auto-correlation observed.

The fitted distributions allow us to explore the effect of replication on latency under realistic traffic patterns. As the number of servers is not disclosed, we set it to 50, but other values



(a) Baseline utilization: 0.1 (b) Baseline utilization: 0.3

Figure 5.11: Percentile improvement in the case study

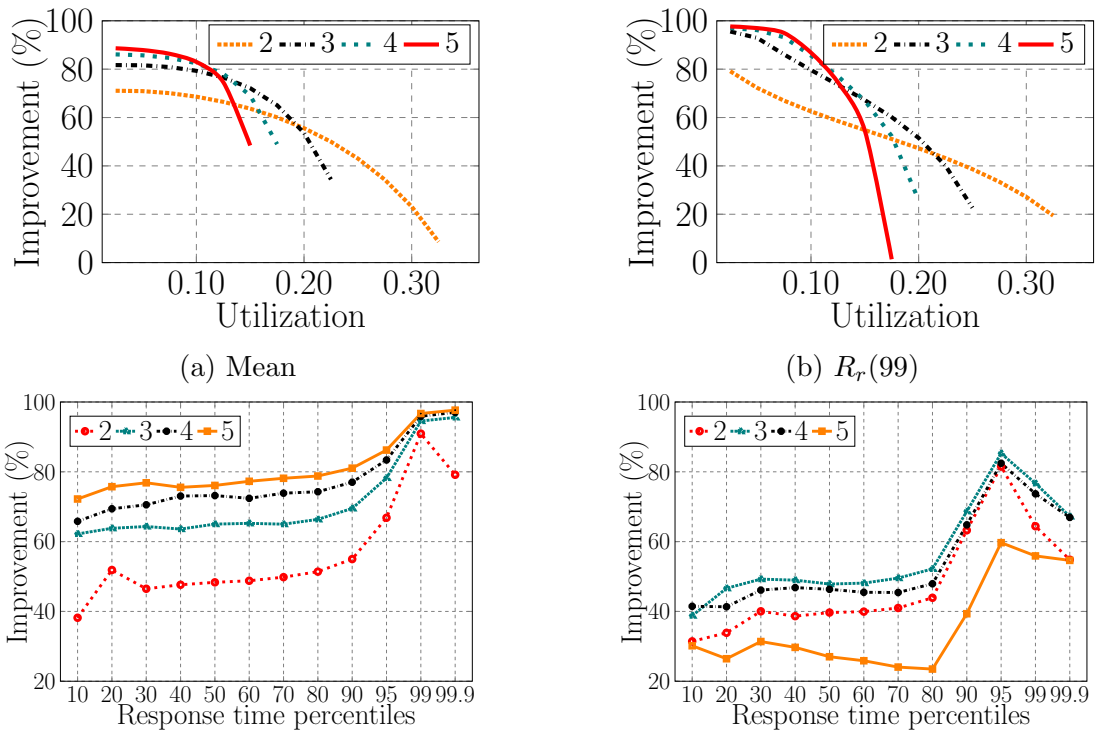
provide similar results. Figure 5.11(a) depicts the relative improvement on response time percentiles when introducing replication under a baseline utilization of 0.1, with grouped-random allocation. The first immediate observation is that the improvement is very different among the percentiles, and it also differs among traces. The log C shows a larger improvement in the tail, while log D shows a stronger improvement for small percentiles. Figure 5.11(a) also shows how the improvement would be evaluated if the empirical service times were modeled with a standard exponential distribution. Surprisingly, both traces show the same improvement across all percentiles, even though the arrival processes are very different. More importantly, the exponential assumption hides the major differences in the lower percentiles and underestimates the benefits of replication. In Figure 5.11(b) we increase the baseline utilization to 0.3 and observe that replication significantly improves the tail for both traces, although this improvement is more significant for log C. Also, the improvement of log C dominates that of log D for most of the distribution, while under a 0.1 utilization this only occurs for percentiles 60 and above. These trends are quite different from the exponential assumption, which leads to the conclusion that replication would be harmful for the tail. Explicitly considering the service time variability shows that it is actually the tail that can benefit the most from replication.

Beyond two replicas

We now consider the case with more than one additional replica. The introduction of replicas increases the utilization by a factor r compared to the baseline utilization, thus limiting the introduction of more than one replica to low-utilization cases. Focusing on random allocation as an example, Figures 5.12(a) and 5.12(b) show that the addition of more than one replica reduces the response time mean and 99^{th} percentile, although under more limited baseline utilizations. In fact, the improvement in the 99^{th} percentile with 5 replicas can be above 95% for a utilization under 0.1, but this improvement decreases very fast and becomes worse than the two-replica case for utilizations above 0.15. This is further illustrated in Figure 5.12(c), which shows the improvement across many percentiles under a very low utilization. Here each additional replica improves every percentile, with the main gains obtained in the tail. However, under a 0.15 utilization, Figure 5.12(d) shows that the improvement obtained with the third and fourth replica is very limited across all percentiles, compared to the two-replica scenario, while the addition of a fifth replica is actually harmful. Thus, except for scenarios with very low utilizations, the main benefits of replication can be obtained with a single additional replica. Additional replicas can provide further gains but only under very low utilizations.

5.2 The centralized set-up: replication without canceling

In this section we consider the *centralized* set-up, where the system consists of a central dispatcher and multiple distributed servers. We are interested in evaluating speculative replication without canceling for applications with very short processing times. We introduce a novel service model in this chapter to explicitly consider **correlation** in the processing times of the request replicas, and design an efficient algorithm to parameterize the model from real data. Extensive evaluations on a MATLAB benchmark and a three-tier web application (MediaWiki) show remarkable accuracy, e.g., 7% (4%) average error on the 99^{th} percentile response time for the benchmark (respectively, MediaWiki), the requests of which execute in the order of seconds



(a) Mean (b) $R_r(99)$

(c) Baseline utilization: 0.025 (d) Baseline utilization: 0.15

Figure 5.12: Improvement with $r > 2$ replicas

(respectively, milliseconds). Insights into optimal replication levels are thereby gained from this precise quantitative analysis, under a wide variety of system scenarios. The ubiquity of cloud computing has enabled many service providers to exploit almost unlimited resources in a pay-as-you-go model. While this model offers many advantages, it also poses new challenges, some of them related to the virtualized nature of the cloud offering. In this section, we are interested here in evaluating the impact of *replication without canceling*, which is asynchronous, and to this end, we derive a suite of stochastic models that accurately predict request response times.

Moreover, through experimentation, we have observed that the processing times of replicas of the same request can be **correlated**. To capture this behavior, we introduce a novel approach that uses correlated hyper-Erlang (CHE) distributions in an extension of our model. At the same time, we apply an efficient fitting method based on the Expectation-Maximization algorithm to parameterize the CHE service model with respect to real data. The MediaWiki experiments demonstrate the ability of the CHE service model to accommodate correlated service times effectively, evident by its accurate estimation of response time percentiles. Based on

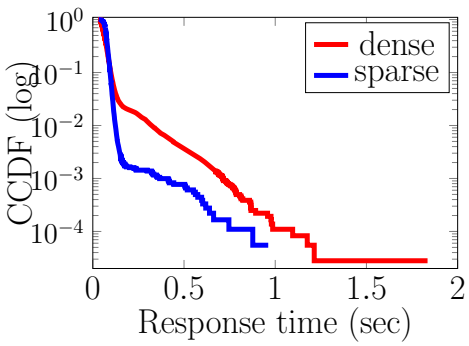


Figure 5.13: Dense vs. sparse setup: response-time CCDF

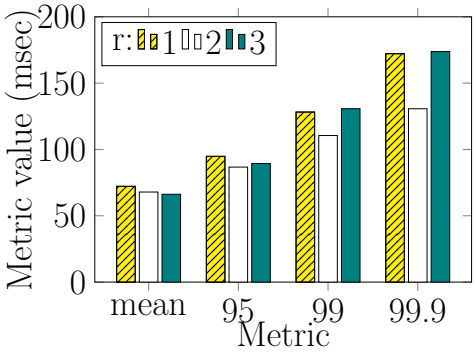


Figure 5.14: Response times under replication.

these models, we derive insights into the design of replication policies, e.g., optimal number of replicas, threshold utilization, etc, for real-world applications.

Most existing works [DB13, SLR16, JSW15] consider the adoption of canceling to limit the additional utilization introduced by the replicas. From an analytical standpoint, the case we consider in this section is more challenging than its counterpart with canceling, where replicas of a request always finish service at the same time, adding a synchronization point that simplifies the system dynamics. A more complex analysis is required to capture the asynchronous behavior in the absence of canceling.

In many applications, requests are fairly homogeneous and their processing times depend largely on the server, leading to a degree of independence between replicas' service times. However, in other applications, processing time depends, at least partially, on the intrinsic details of the request, which impacts all of its replicas. In such cases, therefore, correlation among the replicas of a request becomes significant; whilst commonly ignored [QP15c, VGM+13, GZD+15, QP15b, QP14], it is taken into account in [JSW15]. In contrast, this section considers cases of both independent and correlated processing times among replicas of the same request.

5.2.1 Motivation and system set-up

Employing request replication to mitigate the latency tail poses a number of challenges, which we now illustrate by means of experiments conducted on a real-life web application. We make use of MediaWiki [med15], a standard three-tier web application deployed on a private cloud.

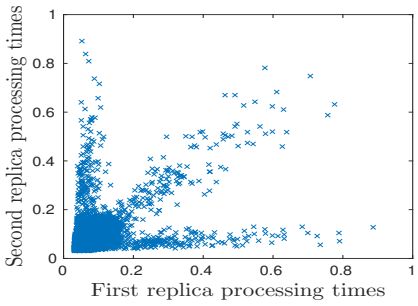


Figure 5.15: Distribution of processing times of two replicas from the same request.

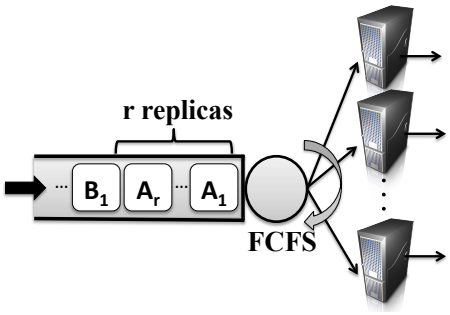


Figure 5.16: Reference model.

Details of the application and its deployment can be found in Section 5.2.6. The cloud deployment exposes the application to large variations in its effective resource capacity, caused by virtual machines (VM) co-located on the same physical machine, a notorious drawback of cloud computing [FJV+12, SDQR10]. This resource capacity variability can cause large variations on the application response times, as illustrated in Figure 5.13. Here we show the CCDF of request response times for two system set-ups: dense, where the six MediaWiki VMs are co-located on three physical servers; and sparse, where each MediaWiki VM is hosted on a separate physical machine. In the densely co-located cloud, the tail can be several times larger than the average response time (73.3 msec). For example, the 95$^{\text{th}}$ and 99$^{\text{th}}$ percentiles are 117.2 and 321.7 msec, respectively, while the 99.9$^{\text{th}}$ percentile is 713.0 msec, which is 9.73× larger than the average. In the sparse case, the mean is similar to that of the co-located case, but the tail is much shorter, with the 99.9$^{\text{th}}$ percentile being only 385.0 msec, which is 5.05× larger than the average.

Figure 5.14 depicts the response time mean and tail percentiles when implementing between 1 and 3 replicas for an arrival rate of 10 requests per second. Clearly, replication is effective in reducing the latency tail, with reductions close to 20% with 2 replicas. However, introducing a third replica hurts the tail even if it improves the mean latency. This highlights the importance of developing analytic models able to compute the latency *distribution* under replication, as those proposed in this chapter, and not just the mean. A common assumption when modeling replication is that the processing times of a set of replicas of the same request are independent. Using measurements from the MediaWiki application with an arrival rate of 10 requests per second and 2 replicas, Figure 5.15 depicts a scatter plot of the processing times of the replicas

Table 5.5: Transition rates for S and $A^{(\text{jump})}$ with PH services

Matrix	Condition	From	To	Rate
S	$n_w \geq 0$	$(\boldsymbol{n}, n_w)$	$(\boldsymbol{n} + \boldsymbol{e}_j - \boldsymbol{e}_i, n_w)$	$n_i S_{\text{rep}}(i,j)$
	$n_w \geq 1$	$(\boldsymbol{n}, n_w)$	$(\boldsymbol{n} + \boldsymbol{e}_j - \boldsymbol{e}_i, n_w - 1)$	$n_i S^*_{\text{rep}}(i)\boldsymbol{s}_{\text{rep}}(j)$
$A^{(\text{jump})}$	$n_w = 0$	$(\boldsymbol{n}, 0)$	$(\boldsymbol{n} + \boldsymbol{e}_j - \boldsymbol{e}_i, r - 1)$	$n_i S^*_{\text{rep}}(i)\boldsymbol{s}_{\text{rep}}(j)$

of each request. Here we observe a large mass along the 45° line, indicating a strong correlation between the replicas' processing times. Also, two other masses close to the axes indicate that in many cases one replica has a long processing time whereas the other has a short one. Thus, the replica processing times are highly varying but not independent, key features that shall be considered in the analytical model proposed in the next sections.

Reference model

Based on the motivating case study presented earlier, our reference model, shown in Figure 5.16, consists of a central dispatcher and c distributed, homogeneous, and independent servers. Requests arrive at the dispatcher and join the next server that becomes available with FCFS scheduling. For each arriving request, $r \geq 1$ replicas asking for the same content are initiated simultaneously and added to the dispatcher. The request is considered complete as soon as one of its r replicas completes service. The remaining $r - 1$ replicas continue their execution without being terminated. The mean replica processing time is $1/\mu$, and we consider two main types of replica processing times, namely, phase-type and correlated hyper-Erlang, to capture the key features of the processing times found in the case study. Our objective is to derive the response-time distribution of requests analytically, for any given replication factor r, number of servers c, and processing-time distribution.

5.2.2 Independent Processing Times

To obtain the *job* response-time distribution, the analysis is divided into two stages: we first obtain the job waiting-time distribution and then find the job service-time distribution.

The waiting-time distribution

To obtain the waiting-time distribution, we observe the queue *only during all-busy periods*, as jobs that arrive during a not-all-busy period see at least one idle server and start service without waiting. In the same vein as in [AM01], we define a bi-variate Markov process $\{X(t), J(t)|t \geq 0\}$, where the *age* $X(t)$ is the total time-in-system of the *youngest* job in service, and the *phase* $J(t) = (D_{\text{all-busy}}(t), A(t))$ holds the phases of both the MAP arrival process $A(t)$ and the service process $D_{\text{all-busy}}(t)$. During an all-busy period, as a replica in service can be in any phase $1 \leq i \leq m_{\text{rep}}$, we define the service state to be $D_{\text{all-busy}}(t) = (n_1(t), \ldots, n_{m_{\text{rep}}}(t), n_w(t))$, where $n_i(t)$ is the number of replicas in service phase i and $n_w(t)$ is the number of replicas of the *youngest job* waiting in the queue, at time t. Thus $D_{\text{all-busy}}(t)$ takes values in the set $N_{\text{all-busy}} = \{(n_1, \ldots, n_i, \ldots, n_{m_{\text{rep}}}, n_w)|n_i \in \{0, \ldots, c\}, \sum_{i=1}^{m_{\text{rep}}} n_i = c, 0 \leq n_w < r\}$, of size $m_{\text{all-busy}}$. As the number of arrival phases is m_a, the total number of phases during the all-busy period is $m = m_a m_{\text{all-busy}}$.

To determine the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution, we first need to find the vector $\boldsymbol{\pi}(x)$, which holds the steady-state density of $\{X(t), J(t)\}$. This has a matrix-exponential representation as shown in Eq. (2.8), which involves to solve Eq. (2.9) to find matrix T first. Table 5.5 summarizes the transition rates of matrices S and $A^{(\text{jump})}$, where $\boldsymbol{n} = (n_1, \ldots, n_{m_{\text{rep}}})$ is the state of the service process before the transition, and $\boldsymbol{e}_i$ is a zero vector with a one in the i-th entry. Here we consider that any of the n_i replicas in service phase i undergoes a transition without service completion with rate $S_{\text{rep}}(i, j)$ or with a service completion with rate $S^*_{\text{rep}}(i)$, where $S^*_{\text{rep}} = -S_{\text{rep}}\mathbf{1}$, allowing a new replica to start service. In the latter case, if no replicas of the youngest job are waiting in queue, a new job starts service and this transition is recorded by the matrix $A^{(\text{jump})}$.

Once T has been found, we obtain $\boldsymbol{\pi}(0)$ to complete the matrix-exponential representation of $\boldsymbol{\pi}(x)$. To find the vector $\boldsymbol{\pi}(0)$, which is the stationary distribution of the phase at the beginning of an all-busy period, we construct a Markov chain by observing the system state every time a new all-busy period starts. To this end we need to connect the not-all-busy with the all-busy periods and follow the evolution of the system during the not-all-busy periods. We thus define

Table 5.6: Transition rates for $S_{\text{(all)-(not-all)}}$ and $S_{\text{not-all-busy}}$

Matrix	From	To	Rate
$S_{\text{(all)-(not-all)}}$	$(\boldsymbol{n}, 0)$	$\boldsymbol{n} - \boldsymbol{e}_i$	$n_i S^*_{\text{rep}}(i)$
$S_{\text{not-all-busy}}$	$\boldsymbol{n}$	$\boldsymbol{n} - \boldsymbol{e}_i + \boldsymbol{e}_j$	$n_i S_{\text{rep}}(i,j)$
	$\boldsymbol{n}$	$\boldsymbol{n} - \boldsymbol{e}_i$	$n_i S^*_{\text{rep}}(i)$

the process $D_{\text{not-all-busy}}(t) = (n_1(t), \ldots, n_{m_{\text{rep}}}(t))$, which keeps track of the number of replicas in service in each phase during the not-all-busy period at time t and takes values in the set $N_{\text{not-all-busy}} = \{(n_1, \ldots, n_{m_{\text{rep}}}) | n_i \in \{0, \ldots, c-1\}, \sum_{i=1}^{m_{\text{rep}}} n_i \leq c-1\}$, which is of size $m_{\text{not-all-busy}}$.

To connect the all-busy period with the not-all-busy period, we introduce the matrix $S_{\text{(all)-(not-all)}}$, which holds the service transition rates (between arrivals) that trigger the initiation of a not-all-busy period. This matrix thus connects the set $N_{\text{all-busy}}$ with the set $N_{\text{not-all-busy}}$, has size $m_{\text{all-busy}} \times m_{\text{not-all-busy}}$, and transition rates as described in the first row of Table 5.6. Note that a transition can initiate a not-all-busy period from a state with zero replicas of the youngest job waiting in the queue, such that a service completion of any replica in service phase i, before an arrival, terminates the all-busy period. Next, we focus on the *not-all-busy period* and introduce the matrix $S_{\text{not-all-busy}}$, which holds the service transition rates between arrivals during the not-all-busy period, as summarized in Table 5.6. Here the second row considers phase transitions without service completions, whereas the third row covers service completions. Also, we adjust the diagonal entries of $S_{\text{not-all-busy}}$ to make it a Markov chain generator.

To track service phase changes due to arrivals during the not-all-busy period, we let $R_{\text{not-all-busy}}$ be an $m_{\text{not-all-busy}} \times m_{\text{not-all-busy}}$ matrix, the $(i,j)^{\text{th}}$ entry of which holds the probability that the service phase just after an arrival is j given that it was i just before. To define this matrix, we introduce the quantity $p(\boldsymbol{n}^w)$, which is the probability that a job starts with w replicas and in phase $\boldsymbol{n}^w = (n_1^w, \ldots, n_{m_{\text{rep}}}^w)$, where n_i^w is the number of replicas that start in phase i, and such that $\sum_{i=1}^{m_{\text{rep}}} n_i^w = w$. As each replica selects its initial phase independently, this probability follows a multi-nomial distribution, thus

$$p(\boldsymbol{n}^w) = w! \prod_{i=1}^{m_{\text{rep}}} \frac{\boldsymbol{s}_{\text{rep}}(i)^{n_i^w}}{n_i^w!}, \tag{5.4}$$

where $\boldsymbol{s}_{\text{rep}}(i)$ is the probability that a replica start service in phase i. We can therefore define

Table 5.7: Transition probabilities for $R_{\text{not-all-busy}}$, $R_{\text{all-busy}}$, and R_{ser}

Matrix	From	To	Prob	Condition
$R_{\text{not-all-busy}}$	$\boldsymbol{n}$	$\boldsymbol{n}+\boldsymbol{n}^r$	$p(\boldsymbol{n}^r)$	$\|\|\boldsymbol{n}\|\|+r<c$
$R_{\text{all-busy}}$	$\boldsymbol{n}$	$(\boldsymbol{n}+\boldsymbol{n}^w, r-w)$	$p(\boldsymbol{n}^w)$	$\|\|\boldsymbol{n}\|\|+r \geq c, w=c-\|\|\boldsymbol{n}\|\|$
R_{ser}	$\boldsymbol{n}$	$(\boldsymbol{n}, \boldsymbol{n}^w)$	$p(\boldsymbol{n}^w)$	$\|\|\boldsymbol{n}\|\|+r \geq c, w=c-\|\|\boldsymbol{n}\|\|$

$R_{\text{not-all-busy}}$ as in Table 5.7, where the arriving job sees more than r idle servers and starts service with all its r replicas, with service phases as described by the vector $\boldsymbol{n}^r$. For clarity we make use of the 1-norm $||\boldsymbol{n}|| = \sum_{i=1}^{m_{\text{rep}}} n_i$, which is the total number of replicas in service. This table also defines a similar matrix $R_{\text{all-busy}}$, of size $m_{\text{not-all-busy}} \times m_{\text{all-busy}}$, which covers the case in which an arrival sees at most r idle servers, triggering the start of an all-busy period. As shown in Table 5.7, in this case the arriving job becomes the youngest in service and starts with $w \leq r$ replicas in service, leaving the remaining $r-w$ waiting in the queue.

We can now find $\boldsymbol{\pi}(0)$ as the solution to the equation

$$\boldsymbol{\pi}(0) = \boldsymbol{\pi}(0) \int_0^\infty \exp(Tu) S^*(u) Q_{\text{not-all-busy}}^{-1} (R_{\text{all-busy}} \otimes D_1) du. \tag{5.5}$$

This equation considers that the age during the all-busy period is u and that a transition triggers the beginning of the not-all-busy period. This occurs according to matrix $S^*(u) = S_{\text{(all)-(not-all)}} \otimes \exp(D_0 u)$, which captures that the next arrival occurs after u time units, such that a service completion triggers the end of the all-busy period. Once the system is in the not-all-busy period, it evolves according to

$$Q_{\text{not-all-busy}} = I_{m_{\text{not-all-busy}}} \otimes D_0 + S_{\text{not-all-busy}} \otimes I_{m_a} + R_{\text{not-all-busy}} \otimes D_1, \tag{5.6}$$

which considers transitions of the arrival process without arrivals, service completions, and arrivals that do not trigger the start of an all-busy-period. Eventually, an arrival causes the process to be absorbed into an all-busy period according to $R_{\text{all-busy}} \otimes D_1$.

To compute the integral in Eq. (5.5), we define

$$P = \int_0^\infty \exp(Tu) S^*(u) du, \tag{5.7}$$

Table 5.8: Transition rates for S_{ser}

From	To	Rate	Condition
$\boldsymbol{n}$	Service completion	$\sum_{i=1}^{m_{\text{rep}}} n_i S^*_{\text{rep}}(i)$	$\|\|\boldsymbol{n}\|\| = r$
$\boldsymbol{n}$	$\boldsymbol{n} + \boldsymbol{e}_j - \boldsymbol{e}_i$	$n_i S_{\text{rep}}(i,j)$	$\|\|\boldsymbol{n}\|\| = r$
$(\boldsymbol{o}, \boldsymbol{n})$	$(\boldsymbol{o} + \boldsymbol{e}_j - \boldsymbol{e}_i, \boldsymbol{n})$	$o_i S_{\text{rep}}(i,j)$	$\|\|\boldsymbol{n}\|\| \leq r-1$
$(\boldsymbol{o}, \boldsymbol{n})$	$(\boldsymbol{o}, \boldsymbol{n} + \boldsymbol{e}_j - \boldsymbol{e}_i)$	$n_i S_{\text{rep}}(i,j)$	$\|\|\boldsymbol{n}\|\| \leq r-1$
$(\boldsymbol{o}, \boldsymbol{n})$	$(\boldsymbol{o} - \boldsymbol{e}_i, \boldsymbol{n} + \boldsymbol{e}_j)$	$o_i S^*_{\text{rep}}(i) \boldsymbol{s}_{\text{rep}}(j)$	$\|\|\boldsymbol{n}\|\| \leq r-2$
$(\boldsymbol{o}, \boldsymbol{n})$	$(\boldsymbol{n} + \boldsymbol{e}_j)$	$o_i S^*_{\text{rep}}(i) \boldsymbol{s}_{\text{rep}}(j)$	$\|\|\boldsymbol{n}\|\| = r-1$
$(\boldsymbol{o}, \boldsymbol{n})$	Service completion	$\sum_{i=1}^{m_{\text{rep}}} n_i S^*_{\text{rep}}(i)$	$\|\|\boldsymbol{n}\|\| \leq r-1$

and integrate P by parts to obtain

$$TP + P(I_{m_{\text{not-all-busy}}} \otimes D_0) = -S_{\text{(all)-(not-all)}} \otimes I_{m_a}.$$

Because T is known, this is a Sylvester matrix equation that can be solved in $O(m_a^3 m^2_{\text{all-busy}}/ m_{\text{not-all-busy}})$ time to find P. As a result, after finding P, determining $\boldsymbol{\pi}(0)$ reduces to solving a linear system with matrix $PQ^{-1}_{\text{not-all-busy}}(R_{\text{all-busy}} \otimes D_1)$, which can be done in $O(m_a^3(m^3_{\text{all-busy}} + m^3_{\text{not-all-busy}} + m^2_{\text{not-all-busy}} m_{\text{all-busy}} + m^2_{\text{all-busy}} m_{\text{not-all-busy}}))$ time.

Note that the stationary distribution $\boldsymbol{\pi}(x)$ exists if $\{X(t), J(t) | t \geq 0\}$ is positive recurrent, which occurs if [Sen89]

$$\varrho = \boldsymbol{\rho} \int_0^\infty u(A^{(\text{jump})} \otimes D_0(u) D_1) du \mathbf{1} > 1, \tag{5.8}$$

where $\boldsymbol{\rho}$ is the invariant probability vector of the matrix $S \otimes I_{m_a} + (A^{(\text{jump})} \otimes D_0^{-1} D_1)$.

Having found $\boldsymbol{\pi}(0)$ and T, we can now obtain the PH representation of the waiting-time distribution following Eq. (2.18), as this coincides with the distribution of the age $X(t)$ just after a downward jump, because that is the age of the job starting service, and thus its waiting time. For the sake of completeness, we recall the computation of γ, by first defining η_0 to be the number of service completions in an all-busy period, and η_1 the number of arrivals in a not-all-busy period, respectively. Their expected values can be obtained following a similar approach as [AM01, Section 7.2]. Thus the probability that a job has to wait is $\gamma = (E[\eta_0] - 1)/(E[\eta_0] - 1 + E[\eta_1])$ since, in a cycle made of one all-busy and one not-all-busy period, $E[\eta_0] - 1$ is the expected number of jobs that have to wait, and $E[\eta_0] - 1 + E[\eta_1]$ is the

expected number of job arrivals.

The service-time distribution

We now move to the second step of the procedure, where we find a PH representation $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ for the job service-time distribution. To determine this distribution, we follow the execution of a tagged job from the time its first replica starts service until one of its replicas completes service. In addition, when not all replicas of the tagged job are in service, we need to keep track of the non-tagged replicas in service, as their completion marks the start of the tagged replicas in the queue. We thus define the service process $D_{\text{ser}}(t)$ as $(n_1(t), \ldots, n_{m_{\text{rep}}}(t))$ when $\sum_{i=1}^{m_{\text{rep}}} n_i(t) = r$ (all tagged replicas in service) and as $\{(o_1(t), \ldots, o_{m_{\text{rep}}}(t)), (n_1(t), \ldots, n_{m_{\text{rep}}}(t))\}$, when $\sum_{i=1}^{m_{\text{rep}}} n_i(t) < r$ (at least one tagged replica in the queue). Here, $n_i(t)$ is the number of tagged replicas in service phase i, whereas o_i is the number of non-tagged replicas in service phase i, at time t. This process thus takes values in the set $N_{\text{ser}} = N_{\text{ser}}^{\text{all}} \cup N_{\text{ser}}^{\text{part}}$, where $N_{\text{ser}}^{\text{all}} = \{(n_1, \ldots, n_{m_{\text{rep}}}) | 0 \leq n_i \leq r, \sum_{i=1}^{m_{\text{rep}}} n_i = r\}$ covers the states where all tagged replicas are in service, and the remaining states are in $N_{\text{ser}}^{\text{part}} = \{(o_1, \ldots, o_{m_{\text{rep}}}), (n_1, \ldots, n_{m_{\text{rep}}}) | 0 \leq o_i < c, 0 \leq n_i < r, 1 \leq \sum_{i=1}^{m_{\text{rep}}} n_i < r, \sum_{i=1}^{m_{\text{rep}}} (o_i + n_i) = c\}$. As N_{ser} is made by the union of two sets, we order it by placing the phases in the set $N_{\text{ser}}^{\text{all}}$ first, and then those in $N_{\text{ser}}^{\text{part}}$.

The transition rates among service phases, which define the sub-generator matrix S_{ser}, can thus be defined as in Table 5.8, where the vectors $\boldsymbol{o} = (o_1, \ldots, o_{m_{\text{rep}}})$ and $\boldsymbol{n} = (n_1, \ldots, n_{m_{\text{rep}}})$ hold the state of all non-tagged and tagged replicas in service before the transition, respectively. The first two rows consider tagged transitions, with and without a service completion, when all tagged replicas are in service. In the remaining rows, at least one tagged replica is still in the queue. Rows three and four cover the cases of transitions without service completion for non-tagged and tagged replicas, respectively. Instead, in rows five and six a non-tagged replica completes service, letting one tagged replica start service. The difference is that in row five at least one tagged replica is left in the queue after the transition, whereas in row six the last tagged replica starts service. The last row considers the service completion of a tagged replica when there still are tagged replicas in the queue.

Having obtained S_{ser}, it remains to determine the vector $\boldsymbol{s}_{\text{ser}}$, which holds the stationary probability that a job starts service in each phase. In other words, $\boldsymbol{s}_{\text{ser}}$ is the distribution of the service phase of the youngest job in service immediately after its service starts. Considering the different conditions in which a job starts service, the $1 \times m_{\text{ser}}$ vector $\boldsymbol{s}_{\text{ser}}$ is given by

$$\boldsymbol{s}_{\text{ser}} = \boldsymbol{s}_{\text{busy}} + \boldsymbol{s}^{\text{all}}_{\text{not-busy}} + \boldsymbol{s}^{\text{part}}_{\text{not-busy}}, \tag{5.9}$$

as we explain in the following.

Busy start If, upon arrival, a job finds all servers busy, its initial phase must consider the state of all non-tagged replicas already in service. This phase is therefore related to the stationary distribution of the service phase after a downward jump in the age process $X(t)$, as these are the time points at which new jobs start service. We know that the joint distribution of the age and the phase is $\boldsymbol{\pi}(x)$. Furthermore, because of the matrix-exponential form, the probability that the age reaches $[x+u, x+u+du)$, given that the age is x and the phase is i, is independent of x. Thus, the probability that level x is visited after a downward jump and that when this occurs service phase $j \in N_{\text{ser}}$ is visited is given by the j^{th} entry of

$$c_1 \int_0^\infty \int_0^\infty \boldsymbol{\pi}(x) \exp(Tu) \left(M \otimes \exp(D_0 u) D_1\right) du dx. \tag{5.10}$$

Here c_1 is a normalizing constant, $\boldsymbol{\pi}(x)\exp(Tu)$ is the probability density of reaching an age in $[x+u, x+u+du)$, and $\exp(D_0u)D_1$ is the probability density of observing a downward jump of size $[u, u+du)$. We also introduce the $m_{\text{all-busy}} \times m_{\text{ser}}$ M matrix, which holds the rates at which a downward jump occurs in service phase $i \in N_{\text{all-busy}}$ and causes the new job to start service in phase $j \in N_{\text{ser}}$. The nonzero entries in matrix M correspond to service-completion rates from service states with $n_w = 0$, where no replicas of the youngest job in service are waiting in the queue, allowing a new job to start service with a single replica. Therefore, a transition from state $(\boldsymbol{n}, 0)$ to state $(\boldsymbol{n} - \boldsymbol{e}_i, \boldsymbol{e}_j)$ occurs with rate $n_i S^*_{\text{rep}}(i) \boldsymbol{s}_{\text{rep}}(j)$. Since $\boldsymbol{\alpha}_{\text{busy}} = -\boldsymbol{\pi}(0)T^{-1}$, we can write Eq. (5.10) as $c_1 \boldsymbol{\alpha}_{\text{busy}} L (M \otimes D_1)$, where $L = \int_0^\infty \exp(Tu)\left(I_{m_s} \otimes \exp(D_0 u)\right) du$ as defined in Eq. (2.10). This matrix has a similar form as P in Eq. (5.7) and can thus be found by solving an associated Sylvester matrix equation. Also, to ensure that (5.10) is stochastic,

we set $c_1^{-1} = \boldsymbol{\alpha}_{\text{busy}} L(D_1 \otimes M)\mathbf{1}$. As γ is the probability that a job has to wait, such a job starts service according to

$$\boldsymbol{s}_{\text{busy}} = \gamma c_1 \boldsymbol{\alpha}_{\text{busy}} L(M \otimes D_1) \tag{5.11}$$

Full start A job arrives during a not-all-busy period with probability $1-\gamma$ and starts service without waiting. Among these arrivals, all but one start service with all their replicas, whereas the last arrival initiates an all-busy period. Recall that η_1 is the number of arrivals in a not-all-busy period, $E[\eta_1]-1$ is then the expected number of jobs among them that find more than r idle servers upon arrival. Thus the probability that a job starts with all its replicas in a not-all-busy period is $p_r = (E[\eta_1]-1)/E[\eta_1]$. When a job finds more than r idle servers, it starts service in state $\boldsymbol{n}^r = (n_1^r, \ldots, n_{m_{\text{rep}}}^r) \in N_{\text{ser}}^{\text{all}}$, where n_i^r is the number of replicas that start service in phase i. We define the vector $\boldsymbol{s}_r$ of size $|N_{\text{ser}}^{\text{all}}|$, with entries $\boldsymbol{s}_r(\boldsymbol{n}^r) = p(\boldsymbol{n}^r)$, with $p(\boldsymbol{n}^r)$ as in Eq. (5.4), for every phase vector $\boldsymbol{n}^r \in N_{\text{ser}}^{\text{all}}$. Thus, in this case, a job starts service with probability vector

$$\boldsymbol{s}_{\text{not-busy}}^{\text{all}} = (1-\gamma)p_r[\boldsymbol{s}_r \quad \mathbf{0}_1^{m_{\text{ser}}-d_r}], \tag{5.12}$$

as the job does not wait with probability $1-\gamma$ and finds more than r idle servers with probability p_r. Here $\mathbf{0}_a^b$ is an $a \times b$ zero matrix. Note that vector $\boldsymbol{s}_r$ is assigned to the first phases as these correspond to the set $N_{\text{ser}}^{\text{all}} \subset N_{\text{ser}}$, where the tagged job has all its replicas in service.

Partial start Finally, if a job finds $1 \leq w \leq r$ idle servers, with probability $(1-\gamma)(1-p_r)$, it starts service with its first w replicas and initiates an all-busy period. To consider this case, we define an $m_{\text{not-all-busy}} \times m_{\text{ser}}$ matrix R_{ser}, the $(i,j)^{\text{th}}$ element of which holds the probability that a job that initiates an all-busy period starts service in phase $j \in N_{\text{ser}}$ given that the service phase was $i \in N_{\text{not-all-busy}}$ just before its arrival. The entries of matrix R_{ser} are shown in Table 5.7, illustrating the start of a new job with w replicas in phase $\boldsymbol{n}^w$ with probability $p(\boldsymbol{n}^w)$ given by Eq. (5.4). The distribution of the service phase at the beginning of an all-busy period is thus given by

$$c_2 \boldsymbol{\pi}(0) \int_0^\infty \exp(Tu) S^*(u) Q_{\text{not-all-busy}}^{-1} (R_{\text{ser}} \otimes D_1) du,$$

where c_2 is a normalizing constant that ensures that the vector is stochastic. The expression considers the probability that the age during an all-busy period is u, and a transition triggers the beginning of a not-all-busy period, during which the system evolves according to the sub-generator $Q_{\text{not-all-busy}}$, until an arrival finds at most r idle servers, initiating an all-busy period and selecting its initial service phase according to $R_{\text{ser}} \otimes D_1$. Note that we can make use of the integral term P, defined in Eq. (5.7), to re-write this expression as $c_2\boldsymbol{\pi}(0)PQ^{-1}_{\text{not-all-busy}}(R_{\text{ser}} \otimes D_1)$, and $c_2^{-1} = \boldsymbol{\pi}(0)PQ^{-1}_{\text{not-all-busy}}(R_{\text{ser}} \otimes D_1)\mathbf{1}$. As a result, vector $\boldsymbol{s}^{\text{part}}_{\text{not-busy}}$ can be obtained as

$$(1-\gamma)(1-p_r)c_2\boldsymbol{\pi}(0)PQ^{-1}_{\text{not-all-busy}}(R_{\text{ser}} \otimes D_1). \tag{5.13}$$

The response-time distribution

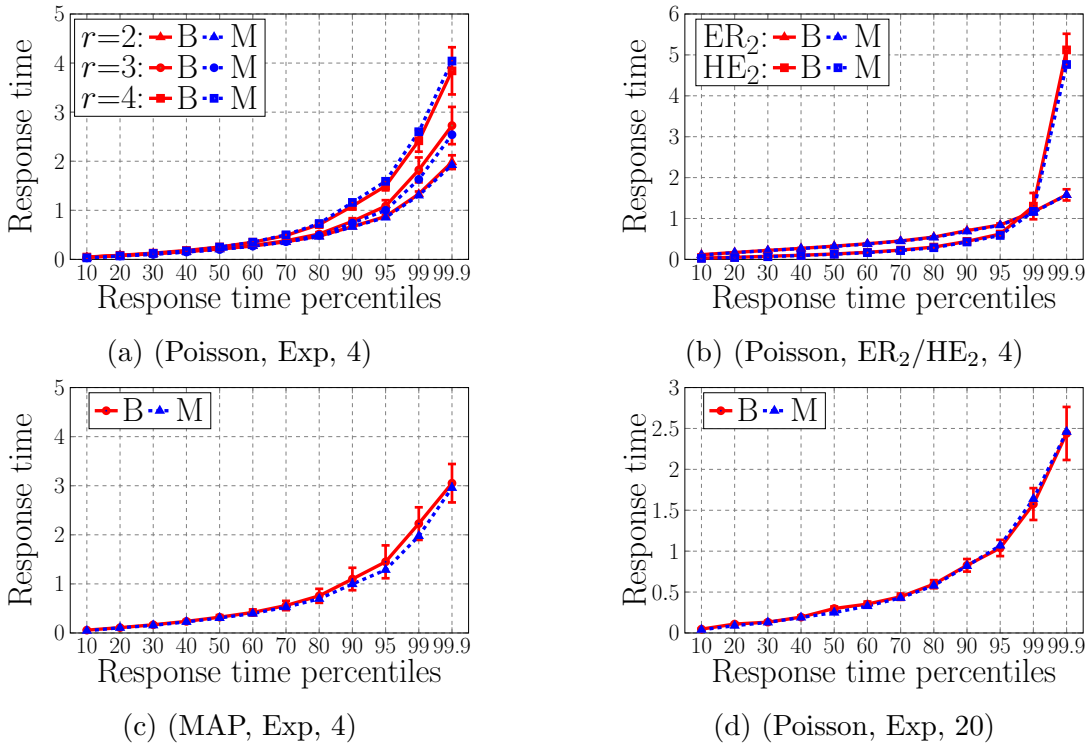
We can now put together the PH representations of waiting and service times to obtain the PH representation $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$ of the response-time distribution as

$$\begin{aligned} \boldsymbol{s}_{\text{res}} &= [\boldsymbol{s}_{\text{wait}}, \quad \boldsymbol{s}^{\text{all}}_{\text{not-busy}} + \boldsymbol{s}^{\text{part}}_{\text{not-busy}}], \\ S_{\text{res}} &= \begin{bmatrix} S_{\text{wait}} & (-S_{\text{wait}}\mathbf{1})\boldsymbol{s}_{\text{busy}}/\gamma \\ \mathbf{0}^{m_a m_{\text{all-busy}}}_{m_{\text{ser}}} & S_{\text{ser}} \end{bmatrix}. \end{aligned} \tag{5.14}$$

Note that to define vector $\boldsymbol{s}_{\text{res}}$ we make use of the $\boldsymbol{s}^{\text{all}}_{\text{not-busy}}$ and $\boldsymbol{s}^{\text{part}}_{\text{not-busy}}$ vectors in Eq. (5.9), which cover the cases where a job starts service without waiting. Other jobs must first undergo a waiting time, starting with vector $\boldsymbol{s}_{\text{wait}}$, after which they start service according to vector $\boldsymbol{s}_{\text{busy}}/\gamma$, where γ is used to re-normalize the vector to be stochastic. With this PH representation, we can directly compute the response-time CDF, percentiles, and moments.

5.2.3 Validation

To test the accuracy of the model introduced in the previous section, we have implemented replication for a simple MATLAB benchmark application, which is executed on a multi-processor host where each processor plays the role of a server in the model. This section describes the



(a) (Poisson, Exp, 4)

(b) (Poisson, ER_2/HE_2, 4)

(c) (MAP, Exp, 4)

(d) (Poisson, Exp, 20)

Figure 5.17: Experimental validation with the MATLAB benchmark on response-time percentiles. Each experiment is labeled by a tuple of (arrival process, processing-time distribution, number of servers). B and M denote results obtained from the benchmark and the model, respectively.

experimental set-up and results.

Experimental set-up

We have implemented replication for a MATLAB benchmark application in which requests arrive to a central dispatcher that keeps a request queue and allocates the first request in the queue to the next available worker. Workers are mapped to independent processors such that the system operation follows the reference model introduced in Section 5.2.1. In particular, we employ the MATLAB blackjack benchmark [Bla15], where each request emulates a game of blackjack that consists of a pre-defined number of blackjack hands. This enables us to modify the distribution of the request execution time by altering the distribution of the number of hands.

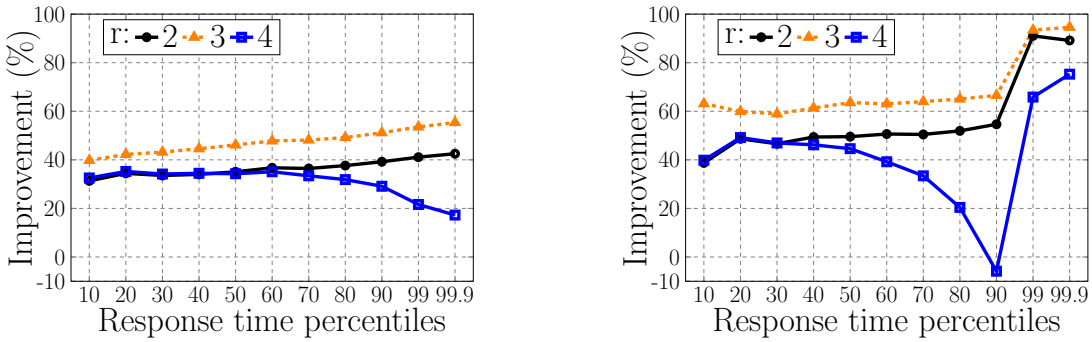
In our set-up requests are generated according to a Poisson or a MAP process. We use MAPs to consider arrivals with high variability and auto-correlation, and we set the SCV of the

inter-arrival times to 10 and the decay rate of the auto-correlation function to 0.5. To set the request execution time we fix the *mean* number of hands to 2000 per request, and generate the number of hands according to three PH distributions, namely, Exp, ER_2 and HE_2. The random variates generated are rounded to obtain an integer number of hands. The SCV for ER_2, Exp, and HE_2 is 0.5, 1, and 10, respectively, showing an increasing degree of variability. We use the methods of [Whi82, DA00] to obtain PH and MAP representations with these moments. In each experiment we run the benchmark ten times, each time executing a total of 5000 requests, and compute the response time percentiles and their 95% confidence intervals.

Performance prediction

We ran the benchmark application on an Intel Core i7-3770 machine with 4 cores running at 3.4 GHz, and 16 GB of memory, launching one worker per core. Running the application without replication we estimate a mean processing time of 0.52 sec, which we use to parameterize the model. Setting the arrival rate to 1 request per second we obtain a utilization of 0.13. We compare the measured and the predicted response times in Figure 5.17, where each experiment is labeled by a tuple of (arrival process, processing-time distribution, number of servers), and we denote benchmark results by B and model results by M in the legend. Under Poisson arrivals and exponentially-distributed processing times, as in Figure 5.17(a), our model shows remarkably good prediction results for replication factors $r = 2$, 3 and 4. For instance, the average error for the $r = 2$ case is only 2.72%. Focusing on the $r = 2$ case, Figure 5.17(b) considers more general request processing-time distributions, ER_2 and HE_2, where we observe that our model is also accurate for systems with non-exponential processing times. Similarly, replacing Poisson arrivals by MAP, Figure 5.17(c) shows that our model predicts results well even under highly varying and auto-correlated arrivals.

We have also deployed the application on a shared cluster made of 4 Intel Xeon E5-4650 machines, with a total of 32 cores running at 2.70 GHz, and 512 GB of memory. Here we deploy 20 workers and set the arrival rate to 4 requests per second. With a measured mean request execution time of 0.71 sec, we achieve a utilization of 0.14. Figure 5.17(d) displays



(a) ER_2 services (b) HE_2 services

Figure 5.18: Percentile improvement.

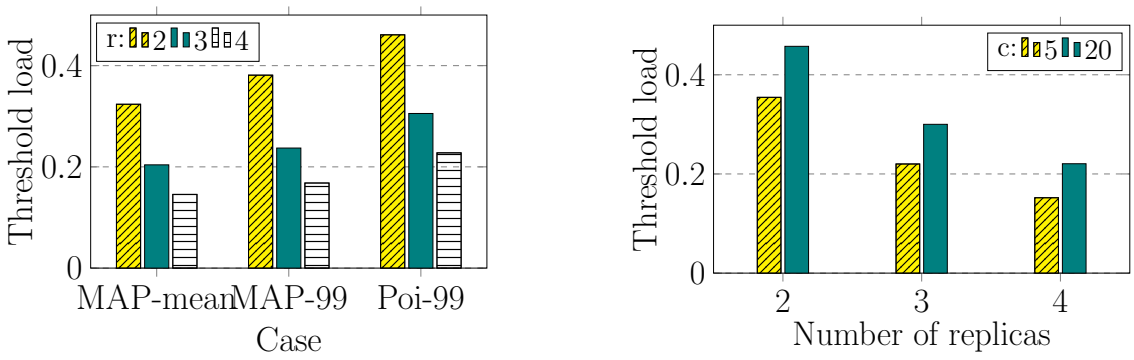
the results for this case, under Poisson arrivals and exponentially distributed processing times, where we observe that our model predicts the entire response-time distribution well, showing a good prediction capability under a high level of parallelism.

5.2.4 Insights regarding replication

The excellent accuracy of the model enables us to explore a large experimental space efficiently and derive insights regarding replication. Particularly, we aim to answer two key questions: (i) when can replication improve the performance and to what extent; and, (ii) what is the maximum baseline utilization for replication to improve the response times compared with the baseline.

The impact of replication

For the results in this section we consider a system with $c = 20$ servers, Poisson arrivals and two processing-time distributions, namely, ER_2 and HE_2, defined in Section 5.2.3. The mean processing time is set to 1 sec, and the arrival rate is set to achieve a baseline utilization of 0.2. Figure 5.18 depicts the relative improvement for each percentile p in the range $\{10, 20, \ldots, 90, 99, 99.9\}$, comparing the response times obtained with replication levels $r = 2$, 3 and 4 against those without replication. For $r = 2$ and 3, the case under ER_2 services, as shown in Figure 5.18(a), has a relatively smooth behavior, with the improvement increasing slightly with the percentile. In contrast, for HE_2 services we observe large peaks in the tail.



(a) Different arrival processes and performance metrics (b) Different number of servers

Figure 5.19: Threshold utilization.

For instance, for $r = 3$ the improvement on the tail is as high as 95%, while the average improvement is around 60%. We have observed a similar behavior in other set-ups, indicating that replication is effective in reducing latency, particularly the tail of the latency distribution. While the improvement increases from $r = 2$ to 3 for all the percentiles, it decreases when r increases to 4, especially for the tail under ER_2 services, and between the 50^{th} and 90^{th} percentiles under HE_2 services. In fact, having 4 replicas is harmful for the 90^{th} percentile under HE_2 services, as the improvement is -5.86%. Thus, care must be taken in evaluating the specific percentiles of interest as the effect of replication is nonuniform over the response-time distribution, to the point that it may benefit some percentiles while hurting others. Also, the service time distribution must be adequately characterized as it has a large impact on the benefits of replication.

The threshold utilization

We now turn to find the *threshold utilization*, that is, the maximum baseline utilization under which replication is beneficial. Here we illustrate our main observations by means of some example set-ups, but we have observed similar behaviors in many other configurations. Note that, for a replication factor r, the baseline utilization is upper bounded by $1/r$, as introducing r replicas in a system with a baseline utilization larger than $1/r$ would make it unstable. To determine the threshold utilization, we rely on the golden section search method [Kie53]. We summarize our key findings in the following.

1. The threshold utilization is nonuniform along the response-time distribution.

Figure 5.19(a), where we assume 20 servers and HE_2 services with a mean processing time of 1 sec, shows how the threshold utilization changes when evaluating different response-time metrics. Comparing the first two cases, which assume MAP arrivals, we observe that the threshold utilization is always larger if we evaluate the 99^{th} percentile rather than the mean. As a result, a decision to replicate based solely on potential gains for the response time tail may actually have a negative effect on the mean. For instance, in this set-up we find that under a utilization of 0.4 the 99^{th} percentile *decreases* by 20% with the introduction of one replica, while the mean *increases* by over 90%. This highlights the importance of explicitly considering the response-time distribution, and the targeted percentiles, when evaluating a replication strategy.

2. The threshold utilization decreases with the variability and auto-correlation of the arrival process.

Replacing MAP arrivals by Poisson, as shown in the third case in Figure 5.19(a), we observe a large increase in the threshold utilization. For instance, for $r = 2$, the threshold utilization increases from 0.38 under MAP arrivals to 0.46 under Poisson arrivals. This is caused by the high burstiness of the MAP process, as replication during bursty periods increases the probability of causing a high utilization, and thus longer delays. Thus, results based on the Poisson assumption, when the actual arrival process displays high variability and/or auto-correlation, can lead to the wrong decision of introducing replication when it is not beneficial.

3. The threshold utilization increases with the number of servers.

Figure 5.19(b) shows the effect of the number of servers on the threshold utilization, using the 95^{th} percentile as the decision-making metric, under Poisson arrivals and Exp services with mean service rate $\mu = 1$. Clearly, the threshold of the 5-server case is smaller than that of the 20-server case. In particular, when introducing 2 replicas, the threshold utilization for the 5-server case is 0.35, whereas it is 0.46 for the 20-server set-up. Thus, more servers provide more flexibility to benefit from replication.

5.2.5 Correlated processing times

As illustrated in Section 5.2.1, replicas of a request display similarities in their processing times. As the model we have introduced assumes independent processing times, these similarities have not been considered. In fact, the independence of processing times is a standard assumption in queueing models, and there are few results for correlated processing times. When such a correlation is considered, as in [GHBDZ10], it is modeled as a general correlation between the processing times of successive requests, whereas here we are interested in correlating the processing times of the replicas of the same job. We thus introduce a model that explicitly captures the correlation among replica processing times and, importantly, fits within the analysis framework defined in Section 5.2.2.

To model correlated processing times within a Markovian model, we introduce the concept of correlated hyper-Erlang (CHE) distributions. Hyper-Erlang distributions [TBT06] model convex combinations of Erlang distributions and are a sub-class of PH distributions. Here we extend this set of distributions to incorporate a dependence between replicas of the same job. Let a CHE distribution be characterized by a set of B branches $\mathcal{B}$, the probability α_i of choosing branch $i \in \mathcal{B}$, the number of exponential phases h_i in branch $i \in \mathcal{B}$, and the rate λ_i in each phase of branch $i \in \mathcal{B}$. These are the parameters that characterize a standard hyper-Erlang distribution. The key difference here is that, while each job is allowed to select a branch independently, all replicas of a job must select the same branch. To model this behavior, we let the first replica of a job select branch $i \in \mathcal{B}$ with probability α_i, and force all other replicas from the same job to select the same branch as the first replica. With these definitions, we extend the model of Section 5.2.2 to handle this modified service process in the following.

The waiting-time distribution

As the replicas of a single job follow a CHE distribution, we let the total number of phases in the PH representation be $m_{\text{rep}} = \sum_{i=1}^{B} h_i$. Next, we redefine the service process during the all-busy periods $D_{\text{all-busy}}(t)$ as $(n_1(t), \ldots, n_{m_{\text{rep}}}(t), n_w(t), b(t))$, where $n_i(t)$ is the number of replicas in

Table 5.9: Transition rates for S and $A^{(\text{jump})}$ - CHE services

Matrix	Condition	From	To	Rate
S	$n_w \geq 0$	$(\boldsymbol{n}, n_w, b)$	$(\boldsymbol{n} - \boldsymbol{e}_i + \boldsymbol{e}_j, n_w, b)$	$n_i S_{\text{rep}}(i,j)$
	$n_w \geq 1$	$(\boldsymbol{n}, n_w, b)$	$(\boldsymbol{n} - \boldsymbol{e}_i + \boldsymbol{e}_b, n_w - 1, b)$	$n_i S^*_{\text{rep}}(i)$
$A^{(\text{jump})}$	$n_w = 0$	$(\boldsymbol{n}, 0, b)$	$(\boldsymbol{n} - \boldsymbol{e}_i + \boldsymbol{e}_j, r-1, j)$	$n_i S^*_{\text{rep}}(i) S_{\text{rep}}(j)$

Table 5.10: Transition rates for $S_{\text{not-all-busy}}$ - CHE services

From	To	Rate
$\boldsymbol{n}$	$\boldsymbol{n} - \boldsymbol{e}_i + \boldsymbol{e}_j$	$n_i S_{\text{R}}(i,j)$
$\boldsymbol{n}$	$\boldsymbol{n} - \boldsymbol{e}_i$	$n_i S^*_{\text{R}}(i)$

service phase i, for $1 \leq i \leq m_{\text{rep}}$, $n_w(t)$ is the number of replicas of the youngest job waiting in the queue, and $b(t)$ is the service branch selected by the youngest job in service at time t. This process takes values in the set $N_{\text{all-busy}} = \{(n_1, \ldots, n_{m_{\text{rep}}}, n_w, b) | n_i \in \{0, \ldots, c\}, \sum_{i=1}^{m_{\text{rep}}} n_i = c, 0 \leq n_w < r, b \in \mathcal{B}\}$.

To describe the evolution of this process, we re-define the matrices S and $A^{(\text{jump})}$ as summarized in Table 5.9. The key difference here is that whenever a new *job* starts service, with rates recorded in matrix $A^{(\text{jump})}$, it chooses a branch j with probability $S_{\text{rep}}(j)$, and this choice is recorded in the last entry of the state descriptor. Thus, when other replicas of the same job start service, with rates in matrix S, they choose the same branch. With these matrices, we can solve Eq. (2.9) to determine the matrix T of the age process.

To determine $\boldsymbol{\pi}(0)$, we again need to consider the evolution of the system during the not-all-busy period. However, during the not-all-busy periods there is no need to keep track of the branch selected by the youngest job in service as this information is only needed when this job has replicas waiting in the queue. We can therefore rely on the service process $D_{\text{not-all-busy}}(t)$ as defined in Section 5.2.2 and on the $S_{\text{not-all-busy}}$ matrix in Table 5.10 to describe the phase process evolution due to service completions. Also, a transition that initiates a not-all-busy period ignores the branch selected by the youngest job in service, as captured by the matrix $S_{\text{(all)-(not-all)}}$ in Table 5.11. In the case of arrivals, the matrix $R_{\text{not-all-busy}}$, shown in Table 5.12, requires a modification to ensure that all replicas of a new job start service in the same phase i with probability $S_{\text{rep}}(i)$. When a job arrives during a not-all-busy period and finds $w \leq r$ idle servers, it initiates an all-busy period, with all its first w replicas starting service in phase

Table 5.11: Transition rates for $S_{\text{(all)-(not-all)}}$ - CHE services

From	To	Rate
$(\boldsymbol{n}, 0, b)$	$(\boldsymbol{n} - \boldsymbol{e}_i)$	$n_i S^*_{\text{rep}}(i)$

Table 5.12: Transition probabilities for $R_{\text{not-all-busy}}$, $R_{\text{all-busy}}$, and R_{ser} - CHE services

Matrix	From	To	Probability	Condition
$R_{\text{not-all-busy}}$	$\boldsymbol{n}$	$\boldsymbol{n} + r\boldsymbol{e}_i$	$S_{\text{rep}}(i)$	$\|\|\boldsymbol{n}\|\| + r < c$
$R_{\text{all-busy}}$	$\boldsymbol{n}$	$(\boldsymbol{n} + w\boldsymbol{e}_i, r-w, i)$	$S_{\text{rep}}(i)$	$\|\|\boldsymbol{n}\|\| + r \geq c, w = \|\|\boldsymbol{n}\|\| + r - c$
R_{ser}	$\boldsymbol{n}$	$(\boldsymbol{n}, w\boldsymbol{e}_i, i)$	$S_{\text{rep}}(i)$	$\|\|\boldsymbol{n}\|\| + r \geq c, w = \|\|\boldsymbol{n}\|\| + r - c$

i with probability $S_{\text{rep}}(i)$, as recorded in matrix $R_{\text{all-busy}}$ in Table 5.12. Here we also keep the selected branch in the last entry of the state descriptor so that the remaining $r - w$ replicas of the new job select the same branch when they start service. With these matrices, we can proceed as done in Section 5.2.2 to obtain the waiting-time distribution.

The service-time distribution

The service-time process requires a similar adaptation to keep track of the service branch selected by the tagged job when some of its replicas are waiting. We thus define the service process $D_{\text{ser}}(t)$ as $(n_1(t), \ldots, n_{m_{\text{rep}}}(t))$ when $\sum_{i=1}^{m_{\text{rep}}} n_i(t) = r$ (all tagged replicas in service), and as $\{(o_1(t), \ldots, o_{m_{\text{rep}}}(t)),\ (n_1(t), \ldots, n_{m_{\text{rep}}}(t), b(t))\}$ when $\sum_{i=1}^{m_{\text{rep}}} n_i(t) < r$ (at least one tagged replica in the queue). Note that the only difference to the service process in the independent case is that we keep track of the branch $b(t)$ selected by the tagged job when at least one tagged replica is in the queue. As this process is very similar to that in the fully-independent case, Table 5.13 summarizes the main differences in S_{ser} with respect to the one in Table 5.8. In this case, when at least one tagged replica is in the queue and a non-tagged replica completes service, a tagged replica starts service in the same phase b selected by all its siblings.

As in Section 5.2.2, to determine the stationary probability vector $\boldsymbol{s}_{\text{ser}}$ according to which a job starts service in each service phase, we consider three different scenarios and obtain this vector as Eq. (5.9). The key differences are:

Busy start In this case, a job starts service with its first replica, which selects its initial service phase according to S_{rep}, just as in the independent case, thus it starts according to Eq. (5.11).

Table 5.13: Transition rates for S_{ser} - CHE services. Main differences to Table 5.8

From	To	Rate	Condition
$(\boldsymbol{o}, \boldsymbol{n}, b)$	$(\boldsymbol{o} - \boldsymbol{e}_i, \boldsymbol{n} + \boldsymbol{e}_b, b)$	$o_i S^*_{\text{rep}}(i)$	$\|\|\boldsymbol{n}\|\| \leq r - 2$

Full start When a job finds more than r idle servers, it starts service with all its replicas. The main difference here is that the replicas select the same phase. Thus, a job selects phase $r\boldsymbol{e}_i$ with probability $S_{\text{rep}}(i)$. We therefore define a vector $\hat{\boldsymbol{s}}_{\boldsymbol{r}}$ of size $|N^{\text{all}}_{\text{ser}}|$, with entries $\hat{\boldsymbol{s}}_{\boldsymbol{r}}(\boldsymbol{n}^r) = S_{\text{rep}}(i)$, when $\boldsymbol{n}^r = r\boldsymbol{e}_i$, and zero otherwise. We then find $\boldsymbol{s}^{\text{all}}_{\text{not-busy}}$, the initial probability distribution of the service phase in this case, by replacing $\boldsymbol{s}_{\boldsymbol{r}}$ with $\hat{\boldsymbol{s}}_{\boldsymbol{r}}$ in Eq. (5.12).

Partial start Finally, if a job finds $1 \leq w \leq r$ servers idle, it starts with its first w replicas, and initiates an all-busy period. The difference to the fully-independent case is that all w replicas select the same phase. We can thus find the distribution of the initial service phase $\boldsymbol{s}^{\text{part}}_{\text{not-busy}}$ as in Eq. (5.13), but adjust the matrix R_{ser}. Table 5.12 shows this modification, where all w replicas start service in phase i with probability $S_{\text{rep}}(i)$, and the selected phase is recorded in the state descriptor to be used when the remaining replicas of the same job start service.

Fitting correlated processing times

The CHE service-time model introduced in Section 5.2.5 allows the r replicas of a job to be correlated in the sense that all of them select the same Erlang branch of the CHE distribution. While fitting methods exists for general PH distributions [ANO96] and some of its sub-classes [KSH03, TBT06], no such method exist for the CHE service model since this is the first time this model is proposed. A maximum-likelihood method, based on the Expectation Maximization (EM) algorithm [DLR77], proposed by Dr. Juan F. Pérez, is developed to parameterize the CHE service model, as described in Appendix A.1.4.

5.2.6 Evaluation on MediaWiki

In this section, we evaluate the capability of the proposed model to predict the response-time distribution of a real-life application, MediaWiki, and show how our proposed analysis can

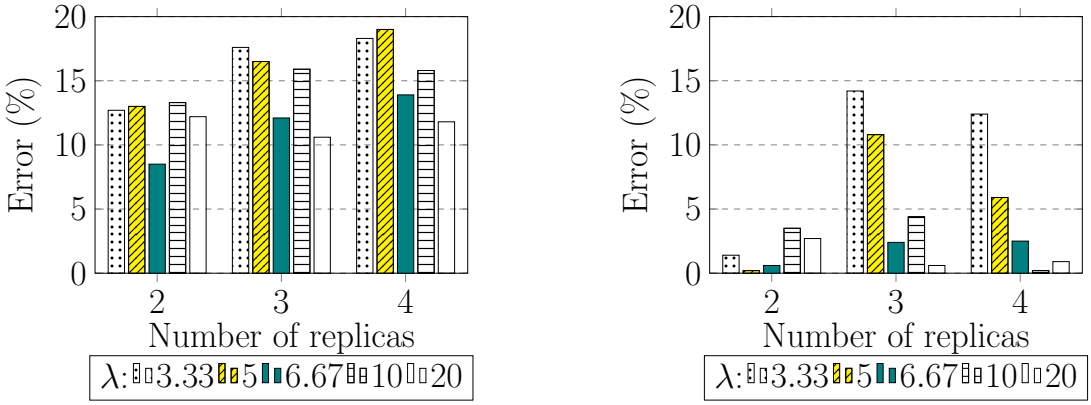
guide the selection of optimal replication factors. We first present the experimental set-up, followed by extensive experiments.

Set-up

Our private cloud testbed is composed of eight identical physical servers, seven used to run the experiments and one used as experiment orchestrator and repository. Each server is equipped with 32 cores, 128 GB DDR4 RAM, six 1-TB solid state disks in RAID5, and two 10-Gigabit Ethernet adapters. We use MediaWiki, the open source platform used to run the Wikipedia website, as a representative application in the cloud [med15]. MediaWiki is a latency-sensitive three-tier web application composed of Apache (v2.4.7) plus PHP (v5.5.9) as application server frontend, Memcached (v1.4.14) as in-memory key-value store and MySQL (v5.5.40) as database backend. In addition, we use an in-house dispatcher written in Go, which replicates and distributes HTTP requests. We deploy a MediaWiki cluster consisting of seven VMs configured with two virtual CPUs and 4 GB of RAM. Six VMs run a complete stack of all three tiers, and one VM is used as the dispatcher. We deploy the cluster in two set-ups: (i) dense, six VMs on three physical servers, each server holding two VMs; and (ii) sparse, each VM hosted on a separate physical server. The average processing time per request for each MediaWiki VM is $1/\mu = 72.98$ msec. Requests are generated with httperf [htt15], an open-loop workload generator. Due to the space limit, we only present the results for the dense set-up in the following.

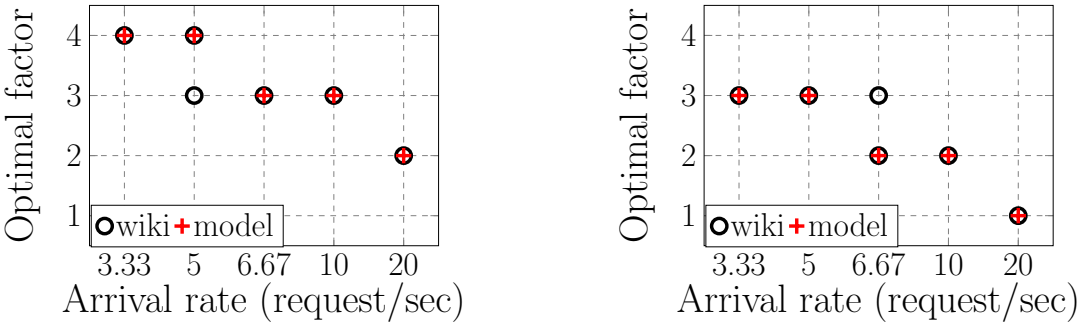
Upon receiving a request, the dispatcher immediately initiates r replicas of the same request and dispatches the replicas to any available VM, with FCFS scheduling. The dispatcher ensures that the maximum number of replicas at each VM is one and that the outstanding requests all wait at the dispatcher. We note that this is a degenerated set-up, whereas the more general set-up immediately forwards the requests to VMs, each of which can hold multiple requests locally.

To emulate performance variability in the cloud, particularly the public cloud, we artificially spawn neighboring workloads following Poisson arrivals and exponential run times. The spe-



(a) Mean response time (b) 99th percentile

Figure 5.20: Experimental validation with the MediaWiki.



(a) Mean response time (b) 99th percentile

Figure 5.21: Optimal replication factor: measurement vs. model.

cific neighboring workload used is fluidanimate, a CPU-intensive benchmark from PARSEC 3.0 [BL11]. Each MediaWiki is co-located with such a neighbor, and we keep the average active time of neighboring workloads around 50% of the MediaWiki experiment time using a mean inter-arrival time of 60 sec and a mean runtime of 30 sec.

Results

We validate our proposed analysis against seven load scenarios, considering Poisson arrivals with mean arrival rates λ of 3.33, 5, 6.67, 10, 20, 30 and 40 requests per second, thus achieving a baseline load of 0.04, 0.06, 0.08, 0.12, 0.24, 0.36 and 0.48 without replication, respectively. For each combination of arrival rate and replication factor, we collect mean and 99$^{\text{th}}$ percentile across 50000 requests resulting in over 100 hours of experiment time. For each arrival rate and replication factor, we parameterize the model from the trace using a CHE distribution with 10 phases and 2 branches. Figure 5.20 summarizes the relative percentage error between

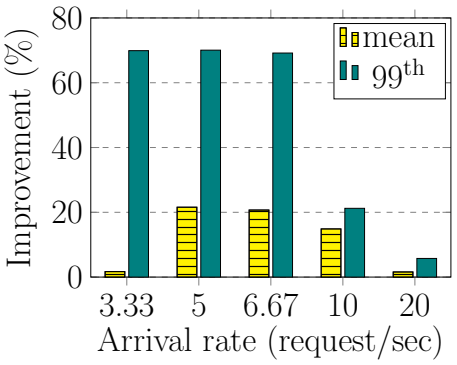


Figure 5.22: Improvement using optimal factor r^*.

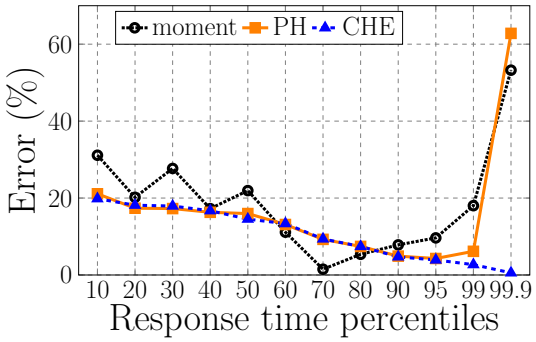
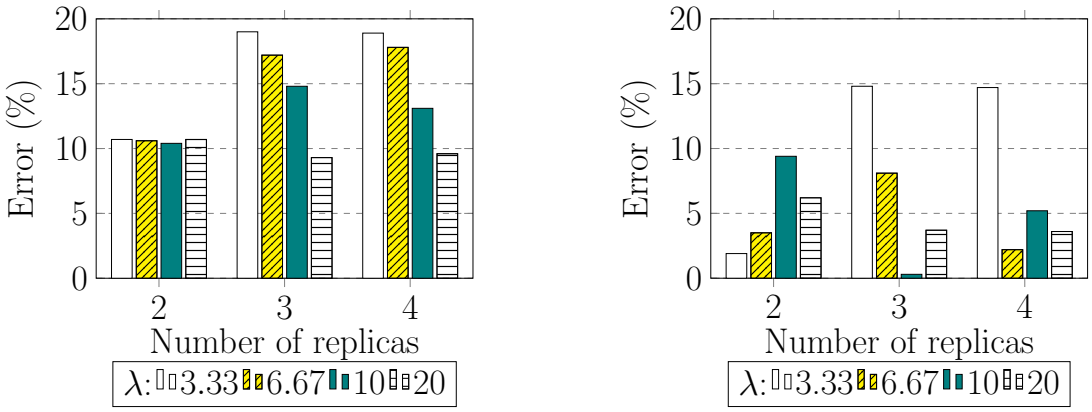


Figure 5.23: Prediction errors using different fitting methods.

the analytical results and the measurements, focusing on the response time mean and 99$^{\text{th}}$ percentile. Here we omit the results for $\lambda = 30$ and 40 as they do not benefit from replication. The errors for these two cases are 11.5% and 9.9% for the mean, and 1.2% and 0.1% for the 99$^{\text{th}}$ percentile, when $r = 2$ for $\lambda = 30$ and 40, respectively. Our model slightly underestimates the mean response time, although the errors are all below 20%. In contrast, we observe that our model performs very well on the 99$^{\text{th}}$ percentile, with most errors under 5%. We note that such fitting results are remarkable given the complexity of the application considered.

A key requirement for a system that implements replication is to determine the optimal replication factor r^*. Figure 5.21(a) and (b) show the empirical and analytical replication factors that minimize the mean and the 99$^{\text{th}}$ percentile, respectively. We find that in certain cases the testbed offers very similar response times under two replication factors. In these cases we allow multiple optimal replication factors if the response times differ by less than 5%. For instance, for an arrival rate of 5, the mean response time for $r = 2$ and 3 is 62.93 and 63.76 msec, respectively, a difference of just 1.32%, thus we consider both replication factors as optimal. From Figure 5.21(a) and (b) we observe that our model identifies the optimal replication factor in all cases considered. As expected, the optimal replication factor decreases with increasing arrival rate, owing to the extra loads introduced by replication. A key observation here is that the optimal replication factors for the 99$^{\text{th}}$ percentile are lower than those for the mean. This confirms our observations in Section 5.2.4 and highlights the importance of considering the targeted percentiles when choosing the replication factor.

When the optimal number of replicas is adopted, Figure 5.22 depicts the improvement in



(a) Mean response time

(b) 99th percentile

Figure 5.24: Prediction errors using $\lambda = 5$.

the mean and the 99th percentile against the set-up without replication. Clearly, replication improves the response times significantly, and has a stronger impact on the tail than on the mean. For instance, when the arrival rate is 5, the improvement is 21.57% on the mean and 70.05% on the 99th percentile. Moreover, for both the mean and the 99th percentile, the improvement is much stronger for low arrival rates as it is more likely for several replicas to execute in parallel, thus the system benefits from selecting the replica that completes first. Instead, under high loads, the extra load introduced by replication leads to longer queueing times, diminishing the benefit.

We now evaluate the introduction of correlation among the replicas' processing times and compare the predictions obtained from three different fitting methods: a moment-matching method [Whi82], an EM method for independent PH distributions [TBT06], and the EM method for CHE distributions introduced in Section 5.2.5. We consider the case with arrival rate 6.67 and replication factor $r = 2$. Figure 5.23 shows the prediction errors on the response time percentiles, $\{10, \ldots, 90, 95, 99, 99.9\}$, compared with the testbed measures. Clearly, the CHE distribution achieves the lowest errors, especially at the tail, where the error on the 99.9th percentile is as low as 0.55%. The independent PH distribution performs well up to the 95th percentile, but fails to capture the tail, which is a key performance metric. The moment-matching method, in contrast, behaves erratically, with a large error on the tail. Further, we compare the ability of the CHE and PH models to capture the 99.9th percentile under different replication factors. With $r = 2$, the 99.9th percentile predicted with the CHE model is 0.3452,

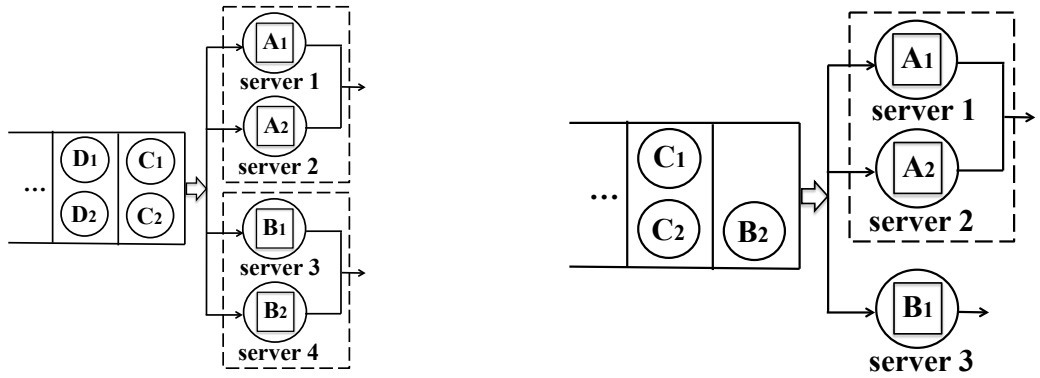
very close to the measurement of 0.3471, while the PH model predicts 0.1291. This difference however diminishes as the replication factor increases, since more replicas benefit more from the resource diversity, weakening the pair-wise correlation. Thus, increasing r to 3, the measured 99.9^{th} percentile is 0.1707, very close to the 0.1667 predicted by the CHE model, while the PH model estimates 0.1415. In terms of optimal replication factor, the CHE model predicts 3 replicas – in agreement with the measurement, whereas the PH model chooses 2 replicas. Again, this observation highlights the advantage of considering the correlation explicitly by fitting the processing times using the CHE model.

Finally, we show that our model can be used to predict the response times under different loads and replication factors. Note that the characteristics of the replica processing-time distribution, in particular, its variability and correlation, differ under different replication factors. Thus, to predict the response times for a given replication factor r, we first fit the replica processing time from a historical trace with the same r, under any load. We then use the fitted distribution to predict the response times for different arrival rates and the same r. As an example, we fit the replica processing times observed for the case with $\lambda = 5$ and $r = 2$, and use them to predict the response times for cases with arrival rates of 3.33, 6.67, 10, and 20, all with $r = 2$. Figure 5.24 shows the errors obtained for the mean and 99^{th} percentile, considering replication factors $r = 2$, 3 and 4. We observe a good prediction performance for all arrival rates, with an average error of 12.61% for the mean and 4.89% for the 99^{th} percentile.

All in all, our analysis can guide the choice of optimal replicas for any response time percentile and a wide range of load scenarios, including different arrival patterns and highly varying and dependent processing times.

5.3 The centralized set-up: replication with canceling

In the previous section, we evaluate replication without canceling for applications with very short processing times, where canceling is hard or even infeasible to implement because of the non-negligible time required to cancel the replicas. In this section, we propose a stochastic



(a) Basic case (b) General case

Figure 5.25: The basic and the general cases

model to evaluate the effect of replication with canceling has on reducing latency with the centralized set-up, considering any number of replicas, general processing and inter-arrival times, and computes the response-time distribution. We then make use of the model to support the selection of the optimal number of replicas.

5.3.1 Reference model

To reduce unnecessary workload, servers communicate updates on the status of their replicas to each other, such that, when a replica completes execution, it immediately cancels all the other replicas in the same job. The removal of all the replicas in a job may require a non-negligible time, which we refer to as the canceling overhead. In the model introduced in the next sections we assume zero canceling overhead, but in Section 5.3.4 we modify the model to explicitly consider overheads of different magnitudes, compared to the job processing times.

For the sake of clarity, we will introduce the model in two steps. First, we consider the *basic case*, where we assume that the number of servers c is a multiple of the job size r, i.e., $c = nr$ for $n \in \mathbb{N}$. In this case the analysis simplifies because all the requests in a job start and complete service at the same time. Figure 5.25(a) illustrates this for the case $c = 4$ and $r = 2$. For instance, the completion of a replica in either job A or B triggers the completion of the corresponding job, and all the replicas in job C start service at the same time. Thereafter, we consider the *general case*, where no restrictions are posed on the number of servers. In this case, a job may start service in either of two states: with all its replicas at the same time (denoted

Table 5.14: Summary of main notations used in this section

Parameter	Definition
c_A	Number of super-servers, $c_A = c/r$
$\text{PH}(\boldsymbol{s}_i, S_i)$	Job service-time PH distribution with m_i service phases $i = job$ for the basic case, $i = F$ for the general case
$A_i^{(\text{jump})}, S_i$	Transition rates with or without a service completion, $i = A$ for the basic case, $i = P$ for the general case
N_i	Set of service phases (cardinality d_i). $i = S$ for all jobs in service, $i = Y$ for the youngest job, $i = N$, initial phases of replicas remaining in the queue

as state A, which is short for *all*); or with only part of its replicas, while the remaining ones wait in the queue (denoted as state P, which is short for *part*). An example of this is depicted in Figure 5.25(b), for the case $c = 3$ and $r = 2$. Here jobs A and B are in service, but job B has one replica waiting in the queue. This replica will only start service once job A completes, and it may not start at all if the replica of job B already in service finishes before job A. The more complex dynamics present in this case will be considered in Section 5.3.3.

In the next sections we propose a model to evaluate the performance of a computing cluster adopting replication with canceling. In addition to the notations summarized in Table 2.1, Table 5.14 summarizes the notations used in this particular model. The model proposed assumes that the replicas of a request are independent. Note that it can also be extended to consider the correlated processing times following similar process in Section 5.2.5.

5.3.2 The basic case

In this section we focus on the basic case, where the number of servers c is a multiple of the number of replicas r, as in Figure 5.25(a). The case without replication, i.e., when $r = 1$, is in fact a MAP/PH/c queue, so we can readily use [AM01] to obtain the waiting-time distribution, and combine it with the service-time distribution to obtain the response-time distribution. The case with replication requires a more detailed analysis to capture the dynamics introduced by the replication and canceling. We assume that the distribution of the *replica* processing times is $\text{PH}(\boldsymbol{s}_{\text{rep}}, S_{\text{rep}})$, and a key step in the following analysis is showing that the *job* processing times also follow a PH distribution, with parameters $(\boldsymbol{s}_{\text{job}}, S_{\text{job}})$ that depend on the overall system

state.

The service-time distribution

As mentioned before, in the basic case all the replicas in a job start service at the same time, and terminate when either of the replicas completes service. As a result we can aggregate the servers into $c_A = c/r$ "super-servers", letting each job be processed by a single super-server. We thus need to describe the job service-time as a PH distribution, which is possible since the job service time is in this case the minimum of the replicas service time, and PH distributions are closed under the minimum operation [LR99]. Since the replicas are homogeneous, instead of recording the service phase of each replica, it suffices to count the number of replicas in each service phase. The service time of a job then follows a PH distribution with representation $(\boldsymbol{s}_{\text{job}}, S_{\text{job}})$, with service phases $N_{\text{job}} = \{\boldsymbol{x}_{\text{job}} = (x_{\text{job}}(1), \ldots, x_{\text{job}}(m)) | x_{\text{job}}(i) \in \{0, \ldots, r\}, \sum_{i=1}^{m} x_{\text{job}}(i) = r\}$, where $x_{\text{job}}(i)$ is the number of replicas in the i-th service phase, for $1 \leq i \leq m$. The number of job service phases is thus

$$m_{\text{job}} = \binom{m+r-1}{r}.$$

The initial distribution of the job service phase $\boldsymbol{s}_{\text{job}}$ follows a multinomial distribution, as each replica picks its initial phase independently. The probability that a job starts service in phase $\boldsymbol{x}_{\text{job}}$ is thus

$$s_{\text{job}}(\boldsymbol{x}_{\text{job}}) = r! \prod_{i=1}^{m} \frac{s_{\text{rep}}(i)^{x_{\text{job}}(i)}}{x_{\text{job}}(i)!}, \quad \boldsymbol{x}_{\text{job}} \in N_{\text{job}}. \tag{5.15}$$

Let $\boldsymbol{e}_s$ be a zero vector with a 1 in its s-th element, such that a job goes from service phase $\boldsymbol{x}_{\text{job}}$ to phase $\boldsymbol{x}_{\text{job}} + \boldsymbol{e}_i - \boldsymbol{e}_j$ when a replica jumps from the j-th phase to the i-th. The elements of the matrix S_{job}, are thus given by $S_{\text{job}}(\boldsymbol{x}_{\text{job}}, \boldsymbol{x}_{\text{job}} + \boldsymbol{e}_i - \boldsymbol{e}_j) = x_{\text{job}}(j) S_{\text{rep}}(j, i)$, as in this case one of the $x_{\text{job}}(j)$ replicas in the j-th service phase jumps to the i-th with rate $S_{\text{rep}}(j, i)$.

As an example, consider the case where the replica service-time distribution has $m = 2$ phases

and sub-generator matrix

$$[S_{\text{rep}}|\boldsymbol{S}^*_{\text{rep}}] = \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \left(\begin{array}{cc|c} -2 & 2 & 0 \\ 0 & -2 & 2 \end{array}\right) \end{array}, \tag{5.16}$$

where C is the absorbing phase, and the vector $\boldsymbol{S}^*_{\text{rep}} = -S_{\text{rep}}\mathbf{1}$ holds the absorbing rates into phase C. Assuming $r = 2$ replicas, the job service phases are $N_{\text{job}} = \{(2,0),(1,1),(0,2)\}$, and its sub-generator matrix S_{job} is

$$[S_{\text{job}}|\boldsymbol{S}^*_{\text{job}}] = \begin{array}{c} \\ (2,0) \\ (1,1) \\ (0,2) \end{array} \begin{array}{c} \begin{array}{ccc} (2,0) & (1,1) & (0,2) \end{array} \\ \left(\begin{array}{ccc|c} -4 & 4 & 0 & 0 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & -4 & 4 \end{array}\right) \end{array},$$

where $\boldsymbol{S}^*_{\text{job}} = -S_{\text{job}}\mathbf{1}$. For instance, in job phase $(2,0)$ either of the 2 replicas in the first phase could jump to the second phase with rate $S_{\text{rep}}(1,2) = 2$, thus $S_{\text{job}}((2,0),(1,1)) = 4$.

The waiting-time distribution

The waiting time of a job is the time period between its arrival and the time its service starts. By aggregating the c servers into $c_A = c/r$ super-servers and looking at the job level, we can compute the waiting-time distribution for the basic case using the approach in [AM01], where we define the bi-variate Markov process $\{X(t), J(t)|t \geq 0\}$, as we have a MAP/PH/c_A queue with PH($\boldsymbol{s}_{\text{job}}, S_{\text{job}}$) service times. Here we detail some of the steps in [AM01] as we will need to treat them in more detail for the general case.

As we aggregate the servers in c_A super-servers, the variable $S(t)$ of the service process records the phases in which the c_A jobs in service are. The set of possible service phases of all the jobs in service is $N_S = \{\boldsymbol{x}_S \in \mathbb{N}^{1\times m_{\text{job}}} | x_S(\boldsymbol{i}) \in \{0,\ldots,c_A\}, \sum_{\boldsymbol{i}\in N_{\text{job}}} x_S(\boldsymbol{i}) = c_A\}$, where $\mathbb{N}^{1\times m_{\text{job}}}$ is a $1 \times m_{\text{job}}$ vector of natural numbers, and $x_S(\boldsymbol{i})$ is the number of jobs in service phase $\boldsymbol{i}$, for

$\boldsymbol{i} \in N_{\text{job}}$. The bold index $\boldsymbol{i}$ here represent the *job* service phase $\boldsymbol{i}$, which is a vector, and is different from the scalar indexes used before for the *replica* service phase. The cardinality of N_S is

$$m_s = \begin{pmatrix} m_{\text{job}} + c_A - 1 \\ c_A \end{pmatrix}.$$

The dimension of the full phase process $J(t) = (A(t), S(t))$ is thus $m_a m_s$, where m_a is the number of arrival phases.

Let the $1 \times m_a m_s$ vector $\boldsymbol{\pi}(x)$ hold the steady-state density of $\{X(t), J(t)\}$. To find $\boldsymbol{\pi}(x)$ it involves to find the matrix T in order to determine the vector $\boldsymbol{\pi}(x)$ as it has a matrix-exponential form such that $\boldsymbol{\pi}(x) = \boldsymbol{\pi}(0)\exp(Tx)$ [AM01]. Further, the matrix T can be found by solving the non-linear integral equation Eq. (2.9), which is

$$T = S_A^{(\text{MAP})} + \int_0^\infty \exp(Tu) A_A^{(\text{MAP})}(u) du,$$

in this case, where $S_A^{(\text{MAP})} = S_A \otimes I_{m_a}$, $A_A^{(\text{MAP})}(u) = A_A^{(\text{jump})} \otimes \exp(D_0 u) D_1$. The elements of S and $A_A^{(\text{jump})}$ can be obtained by considering the transitions from service phase $\boldsymbol{x}_S$ to phase $\boldsymbol{x}_S + \boldsymbol{e_i} - \boldsymbol{e_j}$, for each phase $\boldsymbol{x}_S \in N_S$, and $\boldsymbol{i}, \boldsymbol{j} \in N_{\text{job}}$. Here the vector $\boldsymbol{e_i}$ is a zero vector of size M_{job}, indexed by N_{job}, with a 1 in the entry that corresponds to service phase $\boldsymbol{i}$. Recall that in this case $\boldsymbol{x}_S$ describes the status of *all* the jobs in service. Thus the entry $S(\boldsymbol{x}_S, \boldsymbol{x}_S + \boldsymbol{e_i} - \boldsymbol{e_j})$ is given by $x_S(\boldsymbol{j}) S_{\text{job}}(\boldsymbol{j}, \boldsymbol{i})$, where one of the $x_S(\boldsymbol{j})$ jobs in service phase $\boldsymbol{j}$ jumps to phase $\boldsymbol{i}$ with rate $S_{\text{job}}(\boldsymbol{j}, \boldsymbol{i})$. Also, the element $A^{(\text{jump})}(\boldsymbol{x}_S, \boldsymbol{x}_S + \boldsymbol{e_i} - \boldsymbol{e_j})$ is given by $x_S(\boldsymbol{j}) S^*_{\text{job}}(\boldsymbol{j}) s_{\text{job}}(\boldsymbol{i})$, where one of the $x_S(\boldsymbol{j})$ jobs in service phase $\boldsymbol{j}$ terminates with rate $S^*_{\text{job}}(\boldsymbol{j})$, and a new job starts service in phase $\boldsymbol{i}$ with probability $s_{\text{job}}(\boldsymbol{i})$.

Once T has been found, we need to find the vector $\boldsymbol{\pi}(0)$ and the probability γ that a job finds all servers busy and needs to wait, which we can rely on the similar analysis as Section 5.2.2. We then readily obtain the waiting-time distribution $\text{PH}(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$.

The response-time distribution

Given the PH representations of waiting and service times, we can find the PH representation $(\boldsymbol{s}_{\text{res}}, S_{\text{res}})$ of the response-time distribution following Eq. 2.21 by replacing $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ by $(\boldsymbol{s}_{\text{job}}, S_{\text{job}})$. With this representation we can compute the CDF, percentiles, and moments.

5.3.3 The general case

In this section, we obtain the response-time distribution for the general case, where the number of servers c is not necessarily a multiple of the number of replicas r. In this case, during the all-busy period there are $c_A = \lfloor c/r \rfloor$ jobs in service with all the replicas in service, i.e., in state A, where $\lfloor a \rfloor$ is the largest integer smaller than or equal to a. In addition, the youngest job in service is always in state P, with $p = (c \bmod r)$ replicas in service and the remaining $r - p$ replicas waiting in front of the queue. Instead, during a not-all-busy period, all the jobs in service are in state A. This therefore affects the service-time and waiting-time distributions, as described next.

The service-time distribution

When a job starts service during an all-busy period, it does so in state P as the youngest job in service. We thus introduce $Y(t)$ to keep track of the service phase of the youngest job in service at time t. $Y(t)$ takes values in the set $N_Y = \{\boldsymbol{x}_Y = (x_Y(1), \ldots, x_Y(m)) | x_Y(i) \in \{0, \ldots, p\}, \sum_{i=1}^{m} x_Y(i) = p\}$, where $x_Y(i)$ is the number of replicas in the i-th service phase, for $1 \leq i \leq m$. The cardinality of N_Y is

$$m_Y = \begin{pmatrix} m+p-1 \\ p \end{pmatrix}.$$

The youngest job can complete service if any of its p replicas in service completes service. Otherwise, when one of the c_A jobs in state A completes service, the $r - p$ replicas of the

youngest job waiting in the queue start service, and the job jumps to state A. Thus, for the youngest job in state P, whether it completes service or jumps to state A, depends not only on the service phase of the job itself, but also on the phase of all the jobs in service in state A. Therefore, to describe the job service-time distribution it is essential to keep track of the service phase of all the jobs in service, while the job is in state P. Once it switches from state P to state A, its service time depends solely on the state of its own replicas.

With the previous considerations, we define $\mathrm{PH}(\boldsymbol{s}_F, S_F)$, the job service-time distribution. As a job may be in either state A or P, we partition the sub-generator S_F as

$$S_F = \begin{bmatrix} S_{A,A} & 0_{m_{\text{job}} \times m_F} \\ S_{P,A} & S_{P,P} \end{bmatrix}. \tag{5.17}$$

The $S_{A,A}$ matrix holds the transition rates without a service completion for a job in state A. The service-time distribution of a job in state A was defined in Section 5.3.2, thus $S_{A,A} = S_{\text{job}}$, which is a $m_{\text{job}} \times m_{\text{job}}$ matrix. Similarly, the matrix $S_{P,P}$ holds the transition rates without a service completion when the job is in state P. As stated above, when the job is in state P we need to keep track of $(S(t), Y(t))$, where $Y(t)$ holds the service phase of the job in state P, while $S(t)$ holds the service phase of all the jobs in state A. The set of service phases in this case is thus the product of the two sets, $N_F = N_S \times N_Y$, and its cardinality is $m_F = m_S m_Y$. As as example, consider the case $r = 2$, $m = 2$, and $c = 3$, depicted in Figure 5.25(b). During an all-busy period, one of the two jobs in service is in state A, while the youngest job is in state P with 1 replica in service. The combined phases of jobs in service are $N_S \times N_Y = \{$((1,0,0), (1,0)), ((1,0,0),(0,1)), ((0,1,0),(1,0)),((0,1,0),(0,1)), ((0,0,1),(1,0)), ((0,0,1),(0,1))$\}$.

The elements of the $M_F \times M_F$ matrix $S_{P,P}$ cover two cases. In phase $(\boldsymbol{x}_S, \boldsymbol{x}_Y)$, one of the $\boldsymbol{x}_S(\boldsymbol{j})$ jobs in state A, service phase $\boldsymbol{j}$, may jump to service phase $\boldsymbol{i}$ without a service completion with rate $S_{\text{job}}(\boldsymbol{j}, \boldsymbol{i})$, thus $S_{P,P}((\boldsymbol{x}_S, \boldsymbol{x}_Y), (\boldsymbol{x}_S + \boldsymbol{e_i} - \boldsymbol{e_j}, \boldsymbol{x}_Y)) = \boldsymbol{x}_S(\boldsymbol{j}) S_{\text{job}}(\boldsymbol{j}, \boldsymbol{i})$, for $\boldsymbol{i}, \boldsymbol{j} \in N_{\text{job}}$. Also, in phase $(\boldsymbol{x}_S, \boldsymbol{x}_Y)$, one of the $\boldsymbol{x}_Y(j)$ replicas of the youngest job in the j-th service phase may jump to the i-th without completing service with rate $S_{\text{rep}}(j, i)$, making $S_{P,P}((\boldsymbol{x}_S, \boldsymbol{x}_Y), (\boldsymbol{x}_S, \boldsymbol{x}_Y + \boldsymbol{e}_i - \boldsymbol{e}_j)) = \boldsymbol{x}_Y(j) S_r(j, i)$, for $1 \leq i, j \leq m$.

Finally, the matrix $S_{P,A}$ holds the transition rates with which the youngest job in state P enters state A. When the job in state P jumps to state A, its initial service phase in state A is the combination of the current phase of its p replicas in service, and the starting phase of its remaining replicas in the queue. Let $\boldsymbol{x}_N$ be a $1 \times m$ vector where the entry $x_N(i)$ is the number of replicas starting service in phase i, out of the $r-p$ replicas waiting in the queue. All possible vectors of this type are in the set $N_N = \{\boldsymbol{x}_N = (x_N(1), \ldots, x_N(m)) | x_N(i) \in \{0, \ldots, r-p\}, \sum_{i=1}^{m} x_N(i) = r-p\}$, and the probability $s_N(\boldsymbol{x}_N)$ of the $r-p$ replicas starting in phases $\boldsymbol{x}_N$ can be computed with Eq. (5.15), replacing r by $r-p$, and $\boldsymbol{x}_{\text{job}}$ by $\boldsymbol{x}_N$. Then, in phase $(\boldsymbol{x}_S, \boldsymbol{x}_Y)$, where the youngest job is in phase $\boldsymbol{x}_Y \in N_Y$, if its waiting replicas start service in phase $\boldsymbol{x}_N \in N_N$, the job jumps to service phase $(\boldsymbol{x}_Y + \boldsymbol{x}_N) \in N_{\text{job}}$. Thus the elements of the $m_F \times m_{\text{job}}$ matrix $S_{P,A}$ are given by $S_{P,A}((\boldsymbol{x}_S, \boldsymbol{x}_Y), \boldsymbol{e_j}) = x_S(\boldsymbol{i}) S^*_{\text{job}}(\boldsymbol{i}) s_N(\boldsymbol{x}_N)$, as in this case one of the $x_S(\boldsymbol{i})$ jobs in state A, phase $\boldsymbol{i} \in N_{\text{job}}$ completes service, and the job in state P jumps to state A, phase $\boldsymbol{j} = (\boldsymbol{x}_Y + \boldsymbol{x}_N)$.

To determine the initial probability vector $\boldsymbol{s}_F$ of the job service process, we consider jobs starting service in state A and P separately. During the all-busy period, every job starts service in state P, while in the not-all-busy period, all jobs start service in state A except for the arrival that initiates an all-busy period. As described in Chapter 2, $E[\eta_0]$ is the expected number of job service completions during an all busy period, so it is equal to the expected number of arrivals during such a period, all of which start service in phase P. Also, $E[\eta_1]$ is the expected number of arrivals during a not-all-busy, all of which, but one, start service in phase A. As a result, the probability p_A that a job starts service in state A is given by $(E[\eta_1]-1)/(E[\eta_1]-1+E[\eta_0])$. The initial probability vector $\boldsymbol{s}_F$ is thus

$$\boldsymbol{s}_F = [\boldsymbol{s}_A \quad \boldsymbol{s}_P] = [p_A \boldsymbol{s}_{\text{job}} \quad (1-p_A)\boldsymbol{s}_Y]. \tag{5.18}$$

The waiting-time distribution

As in Section 5.3.2, to find the PH$(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ waiting-time distribution we need to find the matrix T. Apart from changing the identity matrix I_{m_S} to I_{m_F}, we must specify the $m_F \times m_F$

matrices S_P and $A_P^{(\text{jump})}$, which replace S_A and $A_A^{(\text{jump})}$, respectively, in the general case. As the matrix S_P holds the transition rates among the service phases N_F without triggering a service completion during an all-busy period, this is simply equal to $S_{P,P}$. The matrix $A_P^{(\text{jump})}$, instead, holds the transition rates among the phases N_F associated with a service completion during the all-busy period, which can derive from two cases. In the case the job in state P completes service, all its replicas, including the ones waiting in the queue, are canceled immediately, and the next job in the queue starts service in state P. Thus the element $A_P^{(\text{jump})}((\boldsymbol{x}_S, \boldsymbol{x}_Y), (\boldsymbol{x}_S, \boldsymbol{x}_K))$ is given by $x_Y(i)S_r^*(i)s_Y(\boldsymbol{x}_K)$, for $\boldsymbol{x}_S \in N_S$ and $\boldsymbol{x}_Y, \boldsymbol{x}_K \in N_Y$, as any of the $x_Y(i)$ replicas of the youngest job in the i-th service phase completes service with rate $S_{\text{rep}}^*(i)$, and a new job starts service in state P and phase $\boldsymbol{x}_K$ with probability $s_Y(\boldsymbol{x}_K)$. Similarly, when a job in state A completes service, the associated elements of the matrix $A_P^{(\text{jump})}$ are given by $A_P^{(\text{jump})}((\boldsymbol{x}_S, \boldsymbol{x}_Y), (\boldsymbol{x}_S - \boldsymbol{e_i} + \boldsymbol{e_j}, \boldsymbol{x}_K)) = x_S(\boldsymbol{i})S_J^*(\boldsymbol{i})s_N(\boldsymbol{x}_N)s_Y(\boldsymbol{x}_K)$, as any of the $x_S(\boldsymbol{i})$ jobs in state A, phase $\boldsymbol{i} \in N_J$ completes service, and the job that was in state P, jumps to state A, phase $\boldsymbol{j} = (\boldsymbol{x}_Y + \boldsymbol{x}_N)$. In addition, a new job starts service in phase $\boldsymbol{x}_K \in N_Y$. We can now follow the same steps as the basic case to find the waiting-time distribution $\text{PH}(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$.

The response-time distribution

As in Section 5.3.2, the PH representation of the response-time distribution under the general case is given by Eq. 2.21 by replacing $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ by $(\boldsymbol{s}_F, S_F)$.

Utilization

The utilization U is the expected fraction of the time that a server is busy, which can be computed as

$$U = (E[r]\lambda)/(c\mu), \tag{5.19}$$

where $E[r]$ is the expected number of servers used by a job, the mean arrival rate λ is defined in Eq. (2.6), and the job mean service rate μ is given by $\mu = \boldsymbol{s}_{\text{job}}(-S_{\text{job}})^{-1}\mathbf{1}$. In the basic case, all jobs start and complete service in state A, thus $E[r] = r$. Instead, in the general case, jobs may

finish service using either r or p replicas depending on whether they complete service in state A or P. This can be determined by relying on the service-time distribution $\text{PH}(\boldsymbol{s}_F, S_F)$, as defined in (5.17) and (5.18). Let $\boldsymbol{S}^*_{P,C}$ be the vector holding the absorption, or service completion, rates for a job in state P, which is given by $\boldsymbol{S}^*_{P,C} = -(S_{P,A} + S_{P,P})\mathbf{1}$. Thus the probability $p_{P,C}$ that a job completes service in state P, is given by

$$p_{P,C} = \boldsymbol{s}_P(-S_{P,P})^{-1}\boldsymbol{S}^*_{P,C},$$

which captures the path of a job that starts service in state P with initial probability vector $\boldsymbol{s}_P$, and then spends some time in state P before eventually completing service, being absorbed in phase C with probability vector $(-S_{P,P})^{-1}\boldsymbol{S}^*_{P,C}$. Thus the expected number of servers a job uses is given by

$$E[r] = r - p_{P,C}(r - p),$$

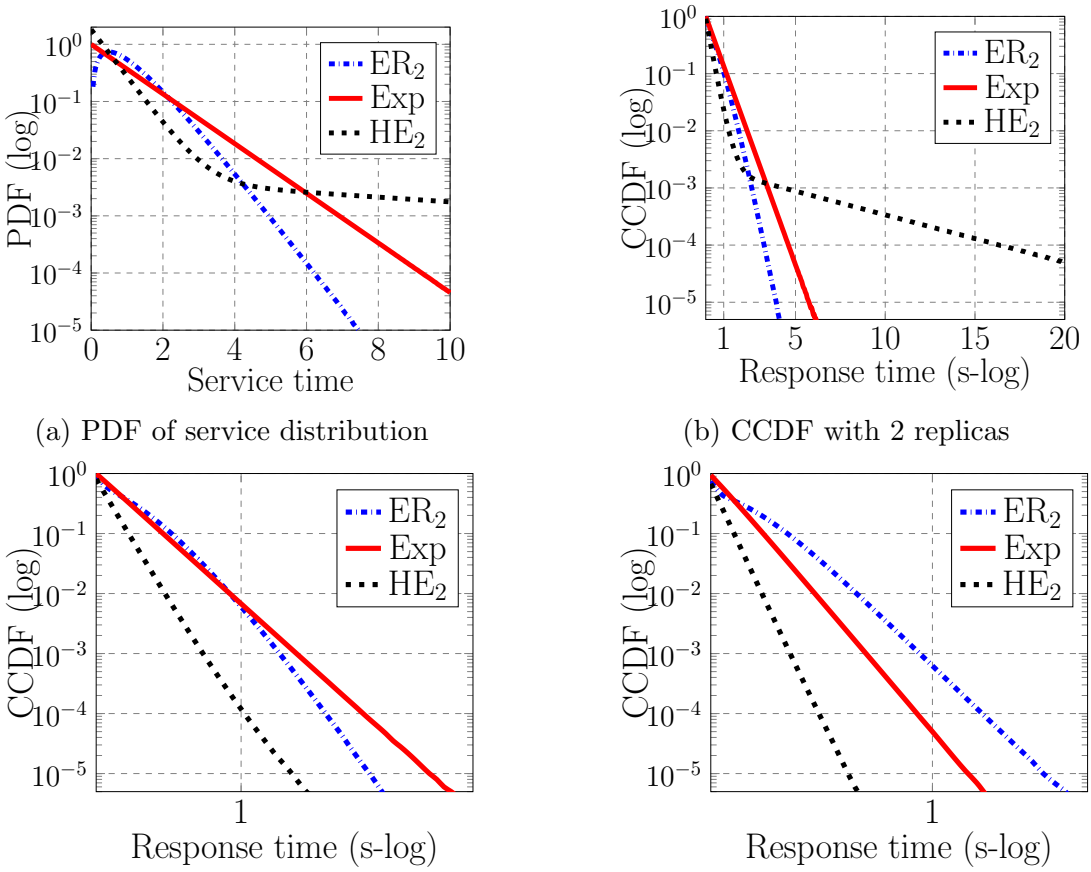
since every job uses r servers, except those that, with probability $p_{P,C}$, use p servers only.

5.3.4 Experimental assessment

We now make use of the proposed model to evaluate the effect of different parameters on the offered response times. Thanks to the flexibility offered by the model, we consider arrival processes with Exp, ER_2, and HE_2 IATs, as well as MAP. For the service-time distribution, we also consider Exp, ER_2 and HE_2 distributions. The mean service rate μ is set to 1, while the arrival rate is set to achieve a target baseline utilization $U = \lambda/c\mu$ for the system without replication.

The effect of the service-time distribution

We now illustrate the impact of the service-time distribution on the system performance, as measured by the response time. We focus on the 99.99^{th} percentile, but similar results hold for other values of the response-time distribution. As a reference, Figure 5.26(a) compares the



(a) PDF of service distribution

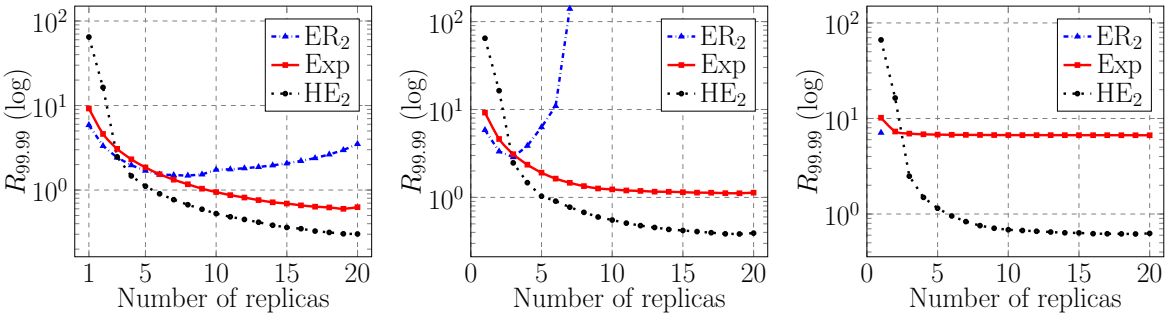
(b) CCDF with 2 replicas

(c) CCDF with 5 replicas

(d) CCDF with 10 replicas

Figure 5.26: Service time PDF and response-time CCDF

probability density functions (PDFs) of the service-time distributions considered, where we see that the HE_2 distribution (SCV = 10) has the longest tail while the ER_2 has the shortest. Also, HE_2 has the highest probability of having short service times while ER_2 has the lowest. Assuming 20 servers, Exp arrivals, a baseline utilization of 0.1, and 2 replicas, Figure 5.26(b) shows how the response-time CCDFs have a similar trend as the service times, with ER_2 and HE_2 having the shortest and longest tails, respectively. When the number of replicas increases to 5, as in Figure 5.26(c), the response time tail shrinks for all service-time distributions, but this change is largest for HE_2 services, which offers the shortest tail. This is further reinforced when the number of replicas increases to 10, as in Figure 5.26(d), where the tail under ER_2 services becomes the longest, and the order is reversed compared to the 2-replica case. The advantage of the HE_2 distribution lies on its larger proportion of small jobs. Since the replication mechanism allows the selection of the first replica that completes service, a larger number of replicas increases the probability that a job benefits from at least one of its replicas having a



(a) Utilization: 0.2 (b) Utilization: 0.5 (c) Utilization: 0.9

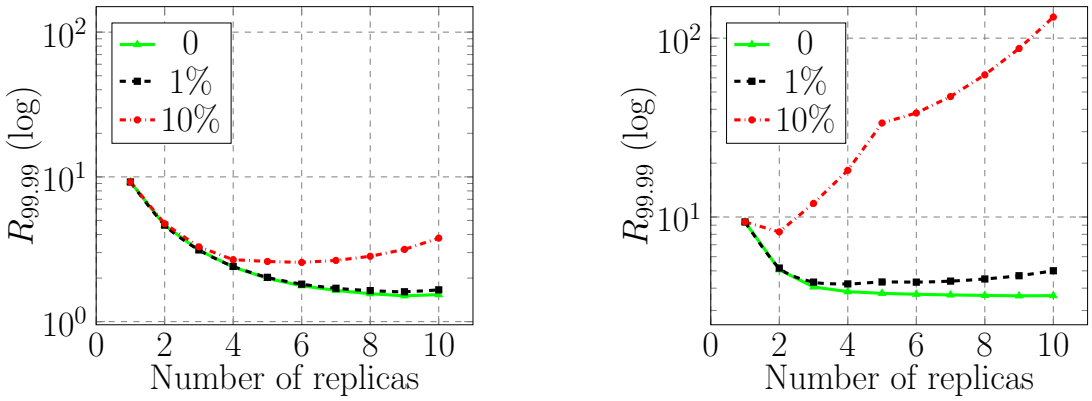
Figure 5.27: Response times under different baseline utilizations and number of replicas

short service time. Replication with canceling is thus more effective if the probability of a short service time is non-negligible. Highly concentrated processing time distributions, like the ER$_2$, offer less chances to gain from replication.

The effect of the number of replicas and the utilization

We now consider a scenario with 20 servers, starting with a relatively low baseline utilization of 0.2. Figure 5.27(a) depicts the $R_{99.99}$ obtained under three service-time distributions, varying the number of replicas between 1 and 20. We observe that, under Exp and HE$_2$ (SCV = 10) services, the $R_{99.99}$ decreases consistently as the number of replicas increases, achieving the lowest $R_{99.99}$ when the number of replicas equals the number of servers. Instead, for ER$_2$ services, the $R_{99.99}$ decreases until the number of replicas reaches 8, and increases afterward. These different behaviors arise from the limited advantage that can be gained from replication under ER$_2$, as service times are highly concentrated. Instead, for Exp and HE$_2$ services, the larger number of replicas increases the probability of having a short job service time. Notice that having more replicas than servers offers no benefit, as at most c replicas can start service at the same time.

When the utilization increases to 0.5, as shown in Figure 5.27(b), the $R_{99.99}$ under ER$_2$ services increases dramatically, achieving a minimum $R_{99.99}$ with 3 replicas, and increasing thereafter until the system becomes unstable with 7 or more replicas. Further, under ER$_2$ services, increasing the utilization to 0.9, as in Figure 5.27(c), makes the system become unstable when introducing only one extra replica. However, under the Exp and HE$_2$ service-time distributions,



(a) Exp, utilization:0.4 (b) MAP, utilization:0.4

Figure 5.28: $R_{99.99}$s under different overhead with Exp services

the $R_{99.99}$ keeps decreasing although the main gain is obtained with the first few replicas. In particular, when the utilization is 0.9, the main reduction in response times is achieved with the first and second replicas, with marginal improvements beyond this number. Replication with canceling can thus be beneficial even under *high utilizations*, as long as the *replica* service-time distribution offers enough variability, as it increases the chances of a short *job* service time.

5.3.5 Introducing the canceling overhead

So far we have assumed that the cost of removing replicas is negligible. However, the canceling operation may require a significant amount of time, compared to the replica processing time. To model the canceling overhead, we introduce an additional phase $\triangle$ in the replica service-time distribution such that, when the replica completes service, it has to spend some time in phase $\triangle$ canceling the remaining replicas of this job before entering the absorbing phase C. For example, assuming a mean canceling overhead of 0.1 in Example (5.16), we have

$$[S_{\text{rep}}|\boldsymbol{S}^*_{\text{rep}}] = \begin{array}{c} \\ 1 \\ 2 \\ \triangle \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & \triangle \end{array} \\ \left(\begin{array}{ccc|c} -2 & 2 & 0 & 0 \\ 0 & -2 & 2 & 0 \\ 0 & 0 & -10 & 10 \end{array}\right) \end{array}$$

where the replica that completes service enters the phase $\triangle$ before entering the absorbing phase C. Figure 5.28(a) shows how the $R_{99.99}$ under 0% and 1% overhead, as a proportion of the replica mean service time, are very similar, decreasing with an increasing number of replicas, assuming Exp arrivals and services, 10 servers and a 0.4 utilization. However, when the overhead increases to 10%, the $R_{99.99}$ reaches a minimum at 6 replicas, and addition replicas degrade the response time. In Figure 5.28(b), we consider the same cases but under MAP arrivals, where we immediately observe larger response times caused by the bursty traffic. The difference is such that, for an overhead of 10%, the $R_{99.99}$ decreases only with the first replica, and even a 1% overhead can make a difference in the number of replicas that achieves the minimum response time.

5.4 Discussion

In this chapter, we have evaluated the effectiveness of request replication to reduce the latency experienced by requests in reliable systems with *general* processing and inter-arrival times, for both the distributed and centralized set-ups. For the distributed set-up, as discussed in Section 5.1, we first consider the small-scale case where $c = r$. Next, we propose an approximate stochastic model to analyze the general case with $c > r$, where the r replicas of each request are allocated to r out of c servers, chosen uniformly at random. We then consider the *centralized* set-up, where we evaluate the systems adopting replication without and with canceling, in Sections 5.2 and 5.3, respectively.

The remarkable accuracy of the proposed models on predicting response-time distributions at a wide range of load scenarios enables us to derive insights on adopting replication. Experimental results show that service time variability plays a key role when evaluating the effect of replication on latency. Highly variable service times provide larger gains, and offer more chances to benefit from replication. Further, we observe that the improvement on latency is not uniform across different percentiles, and the utilization threshold under which replication helps to reduce latency largely depends on the metric targeted. Explicitly considering the

response-time distribution, via the targeted percentiles, is therefore fundamental to evaluate a replication strategy. Also, the introduction of the CHE service model enables us to incorporate the observed correlation among the processing times of replicas of the same request.

Chapter 6

Efficient Approximation for Response-Time Tails in Fork-join Queues

6.1 Introduction

Fork-join queues are natural models for various computer and communications systems that involve parallel multitasking and the splitting and resynchronizing of data, such as parallel computing, query processing in distributed databases, and parallel disk access. In a fork-join queue, each arriving job is split into r tasks, each of which is assigned to one of r parallel processors, or nodes. Once assigned to a node, a task must wait in a queue until its service starts, according to some scheduling policies. After each task is completed, it must wait for all other tasks in the same job to also complete, at which point the job's service terminates. A key performance metric in a fork-join queue is the job response time, although its computation is notoriously difficult due to the synchronization step. In fact, exact solutions for the *mean* response time are only available for fork-join queues with $r = 2$ nodes. A large body of work has therefore been devoted to obtaining approximations for the mean, and, in a few cases, the distribution of the response time. Further, most approximations focus on the case with negative

exponential execution times and Poisson arrivals. Notice that the fork-join queue deals with the *maximum* of node-queue response times, which are not independent due to the synchronized arrivals. This is inherently different from "first to finish" schedules which require a *minimum* value.

In this chapter, we introduce a stochastic model for r-node homogeneous fork-join queues ($r \geq 2$) that focuses on the difference in length between any node-queue and the shortest one, truncating the state space such that the maximum difference is at most a constant C. Whilst most previous methods focus on the *mean* response time, our model is able to evaluate the response-time *distribution*, as well as accommodating phase-type processing times and Markovian arrival processes. In order to tackle scenarios with high loads, which require a large value of C to provide sufficient accuracy, we develop an efficient algorithm using matrix-analytic methods. Tests against simulation show that the proposed model yields accurate results for 2-node fork-join queues. Whilst the matrix-analytic model provides accurate results for small-scale cases, it becomes numerically intractable for large values of r and C. We therefore propose an *Efficient Approximation for Response-Time Tails (EAT)*, which relies on properties of order statistics and extreme values. It stems from an argument that the response-time percentiles of a r-node fork-join queue grow at rate $\mathcal{O}(\ln(r))$, based on the observation that both upper and lower bounds also grow at this rate. EAT has the advantage of being efficient and scalable, requiring only the analytical results for a single-node and 2-node fork-join queues, which we obtain by the preceding matrix-analytic approach. Experimental results show that EAT yields good fits for the tails, even in very large cases with general processing and inter-arrival times. Its scalability and accuracy make the approximation a good candidate to support scheduling, resource provisioning and optimization decisions.

The main contributions of this chapter can be summarized as follows:

- A stochastic model, presented in Section 6.4, to determine the response-time distribution in homogeneous r-node fork-join queues. The model handles phase-type processing times and Markovian arrival processes.
- An approximation based on extreme value theory for fork-join queues with a large number

of parallel servers, which exploits the stochastic model introduced in Section 6.4, to provide accurate estimates of the response-time tail. The approximation approach is described in Section 6.5 and tested in Section 6.6.

Before introducing the stochastic model, we overview related work and provide background definitions.

6.2 Existing work

There is a considerable body of published research on fork-join queues, as recently surveyed in [Tho15]. The complexity of fork-join queues is evident in the paucity of exact results available. With Poisson arrivals and exponential service times, the queue-length distribution for the 2-node queue is derived in [FH84], while [NT88] provides an expression for the mean response time, which has also been obtained in [KA89] for cases with more general inter-arrival times (hyper-exponential) and service times (Erlang). More general cases have been considered only through approximate analysis. In particular, for fork-join queues with Poisson arrivals and exponential service, [NT88] provides an approximation for the mean response time in fork-join queues with r nodes, $2 \leq r \leq 32$. This method is based on the observation that the upper and lower bounds on the mean response time grow at the same rate as a function of the number of nodes r. In this setting, lower and upper bounds on the mean response time [BDVD98] have also been obtained. [LMT94] considered a system with a serial-parallel phase-type arrival process and exponential or Erlang service times, in which a job is split into $1 \leq i \leq r$ tasks. It derived both upper and lower bounds on the expected response time. For general inter-arrival and service-time distributions, approximations for the mean response time in heavy-traffic can be found in [VM94].

Although most previous methods focus on the *mean* response time, a few have considered its *distribution*. An approximation for the case where each node has multiple exponential servers and a Poisson arrival process is considered in [KS04]; it relies on estimating two coefficients through extensive simulations and offers good accuracy for the mean response time when $2 \leq$

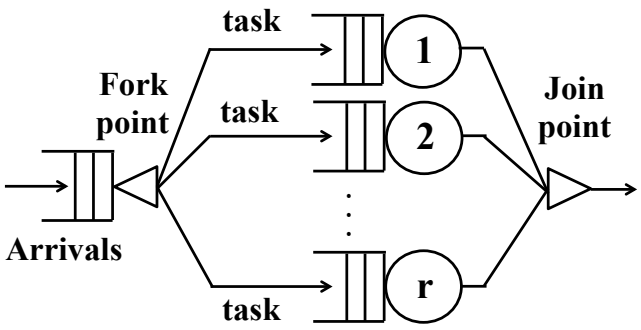


Figure 6.1: Fork-join queue

$r \leq 50$ as well as for the response-time distribution when $r = 2$. A similar approach is proposed in [KS08] for the case that inter-arrival times form a general renewal process, with good accuracy for the response-time distribution when $2 \leq r \leq 50$, assuming Erlang inter-arrival times. In the sequel we introduce a model to compute the response-time distribution in a r-node fork-join queue where service times are phase-type and the arrival process is Markovian (MAP). The model relies on observing the difference in size between any node-queue and the shortest node-queue, and truncating the state space to limit the maximum difference to a constant C. We demonstrate that the model provides accurate results for the response-time distribution. Since the numerics of the model become intractable at large values of r and C, we introduce a method that exploits the results of the model for $r = 2$ to approximate the response-time tails at larger r. Experimental results show that the approximation is accurate even at very large values of r.

6.3 Reference model

We consider a fork-join queue made of r parallel processing resources, each with an associated buffer, as shown in Figure 6.1. Jobs arrive following a MAP(m_a, D_0, D_1), and fork into r tasks, which are sent to the r servers for processing. Tasks in front of a server form a single queue in the order of arrival and receive service when the server becomes available with FCFS scheduling. The r servers are homogeneous, thus the processing time of each task follows a PH$(\boldsymbol{s}_{\text{task}}, S_{\text{task}})$ distribution. For the case of exponential service times, we assume a mean processing time $1/\mu$, such that $\boldsymbol{s}_{\text{task}} = 1$, and $S_{\text{task}} = -\mu$. When a task completes service, it waits at the join point

until all its partners complete execution. When all the tasks of a job complete service, the job service is completed and the job leaves the system. The response time of a job, i.e., the time interval between its arrival and its departure, is thus the maximum of the response times of its tasks. Finally, the system utilization U, i.e., the expected fraction of the time that a server is busy, is simply $U = \lambda/\mu$, where λ is the mean arrival rate, as defined in the previous section, and μ is the task service rate.

6.4 The response-time distribution

In this section, we introduce a stochastic model to estimate the response-time distribution of a parallel job in a r-node fork-join queue, as in Figure 6.1. In the following we pay special attention to the shortest of the parallel queues, and refer to a period during which the server of the shortest queue is busy as an *all-busy* period, and a period during which at least one server is idle as a *not-all-busy* period.

6.4.1 The waiting-time distribution

To determine the waiting-time distribution, we observe the fork-join queue only during the *all-busy* periods, and define a bivariate Markov process $\{X(t), J(t)|t \geq 0\}$. The *age* $X(t)$ keeps track of the total time-in-system of the youngest job in service [AM01], and the *phase* $J(t) = (A(t), D(t))$ holds the joint state of the arrival process $A(t)$ and the service process $D(t)$.

To model the service process of a r-node fork-join queue, we first note that the state of the service process is fully described by the vector $(q_1, q_2, \ldots, q_r)$, where q_i holds the number of tasks in queue i, *either waiting or in service*, for $1 \leq i \leq r$. Since all the servers are identical, we order the queue lengths such that $q_i \leq q_j$ for $i < j$ and $1 \leq i, j \leq r$, making q_1 and q_r the length of the shortest and longest queues, respectively. During the all-busy period the shortest queue length is positive ($q_1 > 0$), while during the not-all-busy period, q_1 is 0 since the shortest queue must be empty. Instead of keeping this description, we focus on the *queue-length*

Table 6.1: Transition rates for S and $A^{(\text{jump})}$

From	To	Rate	Matrix
$(n_0, \ldots, n_{i-1}, n_i, \ldots, n_C)$	$(n_0, \ldots, n_{i-1}+1, n_i-1, \ldots, n_C)$	$n_i\mu$, $i \geq 1$	S
$(n_0, n_1, \ldots, n_{C-1}, n_C)$	$(1, n_0-1, n_1, \ldots, n_{C-2}, n_{C-1}+n_C)$	$n_0\mu$	$A^{(\text{jump})}$

differences with respect to the shortest queue. Since the queues are homogeneous, instead of recording this difference for each queue, it suffices to count the number of queues with the same queue length. We thus model the service process $D(t)$ by keeping track of the number of queues with j more tasks than the shortest queue, $n_j(t)$, for $j \geq 0$. In principle, the difference in length between any queue and the shortest one is unbounded. We thus truncate the state space such that the difference between the longest and the shortest queues is limited to be at most $C < \infty$, such that the $D(t)$ is given by $(n_0(t), n_1(t), \ldots, n_C(t))$. Thus $D(t)$ takes values in the set $N_D = \{(n_0, \ldots, n_C) | \sum_{i=0}^{C} n_i = r, n_i \in \{0, \ldots, r\}\}$. The cardinality of N_D is

$$m_s = \binom{r+C}{r}.$$

The accuracy of this approximation depends on the system parameters. For instance, when the utilization is high, we may expect a higher probability of large differences than under low utilizations, thus requiring a larger value of C to achieve the same accuracy. Section 6.4.5 evaluates the accuracy of this approximation and explores the selection of the limit C.

To determine the PH representation $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$ of the waiting-time distribution, we rely on the stationary distribution $\boldsymbol{\pi}(x)$ of the $(X(t), J(t))$ process, which has a matrix-exponential representation as in Eq. (2.9). Thus we need to find matrices S and $A^{(\text{jump})}$, which are $m_s \times m_s$ matrices that hold the transition rates of the service process associated to transitions without and with a new job starting service, respectively [AM01,Sen89]. The generator of the marginal service phase process during an all-busy period is thus $S + A^{(\text{jump})}$. Notice that the service completion of the task currently in service in any of the n_0 shortest queues triggers a new job to start service, thus corresponding to rates in matrix $A^{(\text{jump})}$; transitions in other queues correspond to rates in matrix S. Table 6.1 summarizes the service transition rates that correspond to matrices S and $A^{(\text{jump})}$. Notice that transitions that lead any of the differences to exceed the limit C are re-assigned to C.

Table 6.2: Transition rates for $S_{\text{not-all-busy}}$

From	To	Rate
$(n_0, \ldots, n_{i-1}, n_i, \ldots, n_C)$	$(n_0, \ldots, n_{i-1}+1, n_i-1, \ldots, n_C)$	$n_i\mu$, $i \geq 1$

Computing $\boldsymbol{\pi}(0)$

To determine the steady state distribution $\boldsymbol{\pi}(0)$ of the phase at the beginning of an all-busy period, it is essential to connect the not-all-busy and the all-busy periods [AM01]. During the not-all-busy period we keep the same description of the phase $J(t) = (A(t), D(t))$, with the service phase $D(t)$ taking values in the set N_D. In this case the shortest queue is empty, and therefore n_0 holds the number of idle servers. Let the $m_s \times m_s$ matrix $S_{\text{not-all-busy}}$ hold the transition rates across service phases, *between arrivals*, during a not-all-busy period, as summarized in Table 6.2.

Since $\boldsymbol{\pi}(0)$ is the distribution of the phase $J(t)$ at the beginning of an all-busy period, it solves Eq. (3.13), as described in Section 2. Then following steps in Section 5.2.2, we obtain the waiting-time distribution.

6.4.2 The service-time distribution

Although the task service-time follows an exponential distribution, the job service-time does not, as it is composed of multiple tasks, and completes service only when all tasks terminate. We now show that the job processing times follow a PH distribution, with parameters $\text{PH}(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$. Let $Y(t)$ be the service state of a tagged job in service at time t, which we define as $Y(t) = (R(t), N(t))$. Here $R(t)$ records the number of tasks in the tagged job that *have not finished service* at time t, thus $R(t) \in \{1, \ldots, r\}$. Similarly to the service phase defined for the waiting-time distribution, the variable $N(t) = (n_0, \ldots, n_C)$ holds the state of the queues, with n_i being the number of queues with i more tasks in queue than the shortest queue. However, here we focus on the queue-lengths in front of the tagged job only, ignoring any job behind the tagged one. This allows us to keep track only of the state of the k queues where tasks in the tagged job have not completed, as the state of the other $r-k$ queues is set to 0 after the tagged tasks

Table 6.3: Transition rates for $S_{\text{(full)-(not-full)}}$ and $S_{\text{not-full}}$

From	To	Rate	Matrix
$(r,(n_0,\ldots,n_i,\ldots,n_C))$	$(r-1,(1,n_0-1,\ldots,n_{C-2},n_{C-1}+n_C))$	$n_0\mu$	$S_{\text{(full)-(not-full)}}$
$(k,(n_0,\ldots,n_{i-1},n_i,\ldots,n_C))$	$(k,(n_0,\ldots,n_{i-1}+1,n_i-1,\ldots,n_C))$	$n_i\mu$ $(i\geq 2)$	$S_{\text{not-full}}$
$(k,(n_0,n_1,\ldots,n_C))$	$(k-1,(n_0+1,n_1-1,\ldots,n_C))$	$n_1\mu$	$S_{\text{not-full}}$

complete. $(R(t),N(t))$ thus takes values in the set $S_{RN}=\{(k,(n_0,\ldots,n_C))|k\in\{1,\ldots,r\},n_i\in\{0,\ldots,k\},\sum_{i=0}^{C}n_i=k\}$.

Therefore, the cardinality of N_{RN} is

$$|N_{RN}|=\binom{r+C}{r}+\sum_{k=1}^{r-1}\binom{k+C}{k}. \tag{6.1}$$

For example, for the case $r=3$ and $C=2$, if only one task has not finished service, thus $R(t)=1$, and $N(t)$ takes values in $\{(2,0,1),(2,1,0)\}$, as the 2 queues where the tagged tasks finished service have queue-length 0, and the other queue can have a queue-length of either 1 or 2.

To obtain a PH representation of the service-time distribution we describe the transitions among the states in N_{RN} with a sub-generator S_{ser}. To this end, we partition N_{RN} into two sets: in the *full* phases none of the tagged tasks have completed service, thus $R(t)=r$; and in the *not-full* phases, at least one tagged task already finished service, thus $1\leq R(t)<r$. Notice that the first set has m_s phases, as defined in Eq. (6.4.1), and letting the number of *not-full* phases be m_{nf}, we see from Eq. (6.1) that $|N_{RN}|=m_s+m_{nf}$. Partitioning the sub-generator S_{ser} according to these subsets, we have

$$S_{\text{ser}}=\begin{bmatrix} S_{\text{full}} & S_{\text{(full)-(not-full)}} \\ 0 & S_{\text{not-full}} \end{bmatrix}\otimes I_{m_a},$$

thus the S_{ser} matrix is of size $m_{\text{ser}}=(m_s+m_{nf})m_a$. Notice that here we have included the arrival phase, such that during service we keep track of the arrival phase at the time when the service started. Although this is not necessary for the service-time distribution on its own, it will become necessary when building the response-time distribution in the next section. Since S_{full} holds the service transition rates without a service completion in the shortest queue, we

have $S_{\text{full}} = S$. The transition rates in $S_{\text{(full)-(not-full)}}$ and $S_{\text{not-full}}$ are summarized in Table 6.3. Notice that the absorbing vector $\boldsymbol{S}^*_{\text{ser}} = -S_{\text{ser}}\mathbf{1}$ has a single nonzero element equal to μ, which corresponds to the phase where the last task completes service. Having obtained S_{ser}, it remains to determine the initial probability vector $\boldsymbol{s}_{\text{ser}}$, which is the stationary probability with which a job starts service in each phase. Following Proposition 3, we have

$$\boldsymbol{s}_{\text{ser}} = [(1-\gamma)\boldsymbol{\pi}(0) + c\gamma\boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a}) \quad \mathbf{0}_{1\times m_{nf}m_a}],$$

where $c^{-1} = \boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a})\mathbf{1}$.

6.4.3 The response-time distribution

Based on the waiting- and service-time distributions, we can adapt Theorem 5.1 to obtain the job response-time distribution, where

$$\boldsymbol{s}_{\text{res}} = \begin{bmatrix} (1-\gamma)\boldsymbol{\pi}(0) & \tilde{\boldsymbol{\alpha}}_{\text{busy}} & \mathbf{0}_{1\times m} \end{bmatrix}, \quad S_{\text{res}} = \begin{bmatrix} S_{\text{ser}} & 0_{m_{\text{ser}}\times m_{\text{ser}}} & 0_{m_{\text{ser}}\times m} \\ 0_{m_{\text{ser}}\times m_{\text{ser}}} & \tilde{S}_{\text{res}} & (-\tilde{S}_{\text{res}}\mathbf{1})P_{s,w} \\ 0_{m\times m_{\text{ser}}} & 0_{m\times m_{\text{ser}}} & S_{\text{wait}} \end{bmatrix}, \tag{6.2}$$

where $\tilde{S}_{\text{res}} = \Delta^{-1}S'_{\text{ser}}\Delta$, Δ is a diagonal matrix such that $\Delta\mathbf{1} = \boldsymbol{\eta}'$, and $\boldsymbol{\eta} = -\boldsymbol{s}_{\text{ser}}S^{-1}_{\text{ser}}$ is the stationary distribution of the phase during service. Also, $\tilde{\boldsymbol{\alpha}}_{\text{busy}} = (-S_{\text{ser}}\mathbf{1})'\Delta$. Finally, $P_{s,w}$ is an $m_{\text{ser}} \times m$ matrix given by

$$P_{s,w} = \begin{bmatrix} \tilde{P}_{s,w} \\ 0_{m_a m_{nf}\times m} \end{bmatrix},$$

where $\tilde{P}_{s,w} = \Gamma^{-1}(T - S \otimes I_{m_a})'\Lambda$, and Γ and Λ are diagonal matrices such that $\Gamma\mathbf{1} = (T - S \otimes I_{m_a})'\Lambda\mathbf{1}$ and $\Lambda\mathbf{1} = \boldsymbol{\alpha}'_{\text{busy}}$. Notice that this result is exact as long as the PH representations of the service and waiting-time distributions are exact. Since we obtain approximate representations for these distributions, the response-time distribution is also approximate.

Table 6.4: Transition rates for S and $A^{(\text{jump})}$ with PH services

From	To	Rate	Matrix
$(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_i, \ldots, \boldsymbol{n}_C)$	$(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_i - \boldsymbol{e}_j + \boldsymbol{e}_k, \ldots, \boldsymbol{n}_C)$	$n_{i,j} S_{\text{task}}(j,k)$, $i \geq 0$	S
$(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_{i-1}, \boldsymbol{n}_i, \ldots, \boldsymbol{n}_C)$	$(\boldsymbol{n}_0, \ldots, \boldsymbol{n}_{i-1} + \boldsymbol{e}_k, \boldsymbol{n}_i - \boldsymbol{e}_j, \ldots, \boldsymbol{n}_C)$	$n_{i,j} S^*_{\text{task}}(j) \boldsymbol{s}_{\text{task}}(k)$, $i \geq 1$	S
$(\boldsymbol{n}_0, \boldsymbol{n}_1, \ldots, \boldsymbol{n}_{C-1}, \boldsymbol{n}_C)$	$(\boldsymbol{e}_k, \boldsymbol{n}_0 - \boldsymbol{e}_j, \boldsymbol{n}_1, \ldots, \boldsymbol{n}_{C-2}, \boldsymbol{n}_{C-1} + \boldsymbol{n}_C)$	$n_{0,j} S^*_{\text{task}}(j) \boldsymbol{s}_{\text{task}}(k)$	$A^{(\text{jump})}$

Table 6.5: Transition rates for $S_{\text{not-all-busy}}$ with PH services

From	To	Rate
$(n_0, \ldots, \boldsymbol{n}_i, \ldots, \boldsymbol{n}_C)$	$(n_0, \ldots, \boldsymbol{n}_i - \boldsymbol{e}_j + \boldsymbol{e}_k, \ldots, \boldsymbol{n}_C)$	$n_{i,j} S_{\text{task}}(j,k)$, $i \geq 1$
$(n_0, \ldots, \boldsymbol{n}_{i-1}, \boldsymbol{n}_i, \ldots, \boldsymbol{n}_C)$	$(n_0, \ldots, \boldsymbol{n}_{i-1} + \boldsymbol{e}_k, \boldsymbol{n}_i - \boldsymbol{e}_j, \ldots, \boldsymbol{n}_C)$	$n_{i,j} S^*_{\text{task}}(j) \boldsymbol{s}_{\text{task}}(k)$, $i \geq 2$
$(n_0, \boldsymbol{n}_1 \ldots, \boldsymbol{n}_C)$	$(n_0 + 1, \boldsymbol{n}_1 - \boldsymbol{e}_j, \ldots, \boldsymbol{n}_C)$	$n_{1,j} S^*_{\text{task}}(j)$

6.4.4 Extension to PH services

The model presented so far has focused on exponential service-times, but it can be extended to the more general case of PH distributions, as we now show. The main difference lies in the service process $D(t)$ as we now need to keep track of the phase of the tasks in service. Thus, assuming the task service-time distribution is $\text{PH}(\boldsymbol{s}_{\text{task}}, S_{\text{task}})$ with m_{task} phases, we define $D^{\text{PH}}(t) = (n_{i,j}(t), 0 \leq i \leq C, 1 \leq j \leq m_{\text{task}})$, where $n_{i,j}(t)$ is the number of queues with queue-length difference i with respect to the shortest queue, and such that the task in service is in phase j, at time t. Thus $D^{\text{PH}}(t)$ takes values in the set $N_D^{\text{PH}} = \{\boldsymbol{n} = (\boldsymbol{n}_1, \ldots, \boldsymbol{n}_C) | \boldsymbol{n}_i = (n_{i,1}, \ldots, n_{i,m_{\text{task}}}), n_{i,j} \in \{0, \ldots, r\}, \sum_{i=0}^{C} \sum_{j=1}^{m_{\text{task}}} n_{i,j} = r\}$.

Similarly, the matrices S and $A^{(\text{jump})}$ need to be re-defined, considering that $A^{(\text{jump})}$ only keeps transition rates associated to the start of a new job service, i.e., service completions in any of the shortest queues. Table 6.4 summarizes the service transition rates that correspond to matrices S and $A^{(\text{jump})}$. The first row in this table considers the case where the task in service goes through a service-phase transition (from phase j to phase k, with $1 \leq j, k \leq m_{\text{task}}$), without completing service, in any of the queues. The second row covers the case of service completions at any queue but the shortest one, which are also kept in S. In this case, the task in service phase j completes, letting a new task start service in phase k, with $1 \leq j, k \leq m_{\text{task}}$. Finally, the third row considers the case where the service completion occurs in any of the shortest queues, which is recorded by matrix $A^{(\text{jump})}$. Here again the task completes service in phase j, and the new task starts in phase k, such that now there is a single shortest queue, which is in phase k.

During the not-all-busy period, a similar modification is necessary, although in this case the service process $D^{\text{PH}}(t)$ needs to reflect that empty queues do not have an associated task in service that requires keeping track of its phase. To differentiate it from the $D^{\text{PH}}(t)$ process during the all-busy period, we label this process D_N^{PH}. The D_N^{PH} process takes values in the set $\bar{N}_D^{\text{PH}} = \{(n_0, \boldsymbol{n}_1, \dots \boldsymbol{n}_C) | \boldsymbol{n}_i = (n_{i,1}, \dots, n_{i,m_{\text{task}}}), 1 \leq i \leq C, n_0 \in \{0, \dots, r\}, n_{i,j} \in \{0, \dots, r\}, n_0 + \sum_{i=1}^{C} \sum_{j=1}^{m_{\text{task}}} n_{i,j} = r\}$. In this case, the first entry of the vector $(n_0, \boldsymbol{n}_1, \dots \boldsymbol{n}_C)$ holds the number of idle servers, while the remaining entries have are the same as for $D^{\text{PH}}(t)$. Table 6.5 summarizes the entries of the matrix $S_{\text{not-all-busy}}$, which are the transition rates across service phases during a not-all-busy period. Here the first row corresponds to phase transitions without a service completion, while the second and third row correspond to service completions. The third row shows how a task in a queue with queue-length one completes service in phase j, thus incrementing the number of idle queues n_0. A final adaptation is needed to compute $\boldsymbol{\pi}(0)$, the distribution of the phase $J(t)$ at the beginning of an all-busy period, which now solves Eq. 5.1. Here the matrix $A^{(\text{jump})}_{\text{not-all-busy}}$ is similar to $A^{(\text{jump})}$ as defined in Table 6.4, but from a state $(\boldsymbol{n}_0, \boldsymbol{n}_1, \dots, \boldsymbol{n}_C) \in N_D^{\text{PH}}$, the transition is to a state $(1, \boldsymbol{n}_1, \dots, \boldsymbol{n}_C) \in \bar{N}_D^{\text{PH}}$ with rate $n_{0,j} S^*_{\text{task}}(j)$. Thus this transition corresponds to a service completion in the shortest queue, which starts a not-all-busy period with one empty queue. On the other hand, the matrix $K^{(\text{start})}$ maps a state $(n_0, \boldsymbol{n}_1, \dots, \boldsymbol{n}_C) \in \bar{N}_D^{\text{PH}}$ into a state $(\boldsymbol{n}_0, \boldsymbol{n}_1, \dots, \boldsymbol{n}_C) \in N_D^{\text{PH}}$ with probability $p_{n_0, \boldsymbol{n}_0}$. Again, the only change is related to the shortest queues, which in this case go from the idle state to an initial phase of service. Once $\boldsymbol{\pi}_0$ is found, the waiting-time distribution is obtained as in the case with exponential services. A similar extension is then needed to obtain the service-time and response-time distributions, which we omit for brevity.

6.4.5 Experimental validation

In this section we show that the proposed model is able to provide accurate results as long as the limit C is large enough. Although the model is defined for any number of parallel queues r, it becomes time consuming and computationally expensive with increasing r, especially in cases with high loads, as these cases require a large value of the limit C to provide sufficient accuracy.

Table 6.6: Approximation error compared with exact results and simulations

Arr	**Ser**	U	**Measure**	**Err(%)**					
				C = 5	**C = 50**	**C = 100**	**C = 500**	**C = 750**	**C = 1000**
Exp	Exp	0.10	mean	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
			$R_2(99)$	0.06	0.06	0.06	0.06	0.06	0.06
		0.50	mean	2.14	<0.01	<0.01	<0.01	<0.01	<0.01
			$R_2(99)$	5.85	0.04	0.04	0.04	0.04	0.04
		0.90	mean	49.30	0.84	0.01	<0.01	<0.01	<0.01
			$R_2(99)$	55.21	3.46	0.05	0.04	0.04	0.04
		0.95	mean	68.19	9.47	0.98	0.03	0.03	0.03
			$R_2(99)$	70.76	20.98	4.17	0.01	0.01	0.01
Exp	HE_2	0.50	mean	33.23	0.42	0.01	0.01	0.01	0.02
			$R_2(99)$	26.81	0.88	0.01	<0.01	<0.01	<0.01
		0.90	mean	83.57	47.28	25.28	0.03	0.03	0.03
			$R_2(99)$	79.44	54.46	38.43	0.11	0.07	0.07
		0.95	mean	91.31	67.96	49.98	3.35	0.38	0.07
			$R_2(99)$	94.26	83.16	59.31	12.60	1.89	0.10
Exp	ER_2	0.5	mean	0.51	<0.01	<0.01	<0.01	<0.01	<0.01
			$R_2(99)$	2.01	0.02	0.02	0.02	0.02	0.02
		0.90	mean	34.11	0.07	<0.01	<0.01	<0.01	<0.01
			$R_2(99)$	40.69	0.23	<0.01	<0.01	<0.01	<0.01
		0.95	mean	54.01	2.60	0.07	0.01	0.01	0.01
			$R_2(99)$	76.50	47.37	45.90	<0.01	<0.01	<0.01
MAP	Exp	0.50	mean	25.43	<0.01	<0.01	<0.01	<0.01	<0.01
			$R_2(99)$	35.23	0.02	<0.01	<0.01	<0.01	<0.01
		0.90	mean	57.32	11.06	3.37	0.07	0.07	0.07
			$R_2(99)$	56.48	13.87	6.24	<0.01	<0.01	<0.01
		0.95	mean	71.05	21.78	11.02	0.23	0.09	0.08
			$R_2(99)$	70.16	22.26	13.26	0.45	0.04	<0.01
MAP	HE_2	0.50	mean	52.35	5.49	0.40	<0.01	<0.01	<0.01
			$R_2(99)$	47.61	11.00	1.24	<0.01	<0.01	<0.01
		0.90	mean	86.40	54.38	36.53	2.10	0.19	0.14
			$R_2(99)$	85.77	57.24	42.63	7.59	1.37	0.08

To handle cases with large r, we propose an approximation approach in Section 6.5, which provides a high degree of accuracy for the response-time tails in fork-join queues with $r > 2$, exploiting the results of the model proposed for the $r = 2$ case. We thus focus on the 2-node fork-join queue, considering different distributions for the arrival and service processes, and different utilization levels. For the service times, we consider Exp, ER_2, and HE_2 distributions. For the arrival process, we consider Exp, and general order-2 MAPs. The Exp and ER_2 distributions have SCV equal to 1 and 0.5, respectively, while for the HE_2 distribution we set it to 10. For MAP arrivals we also set $SCV = 10$, and the decay rate of the auto-correlation function is set to 0.5.

For each setting, the simulations were run for 5,000 times with 200,000 samples each time, from

which we obtain the response-time mean and percentiles, and their 95% confidence intervals. Let the p^{th} percentile of the response time of a r-node fork-join queue be $R_r(p)$. We show in Table 6.6 the errors of the mean response time, and $R_2(99)$ as an example. We also consider different load levels, 0.1, 0.5, 0.9, and 0.95. The case with MAP arrivals, HE_2 services, and load 0.95 is not shown here, as we could not obtain a stable result for the tails via simulation due to the high variability. For increasing values of C, Table 6.6 depicts the relative error obtained with the approximation. For the mean response time we compare against exact methods for 2-node fork-join queues under exponential arrivals and services [NT88, VM94]. For queues with MAP arrivals and PH services, and for the percentiles, we compare against simulation results. With $C = 100$, the relative error is below 1% for most of the test cases, with the exception of HE_2 services, and the cases with 0.95 load, which require a larger C to reach this accuracy. We expect this to be the most challenging case, as the large SCV implies service-times with a long tail, potentially causing large differences among the parallel queues. For instance, with Exp arrivals and HE_2 services, under 0.9 load, the error rate with $C = 100$ and 500 are 38.34% and 0.04%, respectively. In addition, the statistical characteristics of the inter-arrival times have a clear effect on the approximation accuracy, as the error achieved with Exp arrivals is always smaller than the one achieved with MAP arrivals. For instance, for the case with Exp arrivals and services with a 0.95 load, it requires $C = 500$ to reach an error rate lower than 1%, while for the same case but under MAP arrivals, it requires $C = 1000$. This is caused by the high variability and auto-correlation of the arrival process, as this leads to a high probability that short inter-arrival times come in bursts, creating longer queues.

6.5 Approximate analysis

The analytical model introduced in Section 6.4 produces accurate results for 2-node fork-join queues but it becomes time consuming and computationally expensive with increasing r, especially in cases with high loads, which require a large value of the constant C to provide sufficient accuracy. Furthermore, as the value of r increases, simulations must be run for very long times in order to achieve sufficiently accurate values for the response time percentiles, especially in

the tail; again, this is more pronounced at high load. We therefore seek an approximate approach based on the maximum of the response times of the spawned tasks, referred to as the *efficient approximation for tails* (EAT). We start with an approximation for fork-join queues with Poisson arrivals and exponential service times – parallel M/M/1 queues – and then extend to queues with MAP arrivals and PH service. We denote the response time random variable for a r-node fork-join queue by R_r, with p^{th} percentile $R_r(p)$.

6.5.1 Nelson and Tantawi's approximation for mean

Nelson and Tantawi [NT88] derived the mean response-time for a 2-node fork-join queue by decomposing the response-time of each task into the sum of the time R_1 that it spends in its M/M/1 queue and the synchronization delay D spent waiting for its sibling after completion of service, which is non-zero for the task that finishes first and zero for its sibling. Then the total response-time for a job in the 2-node queue is shown to be

$$E[R_2] = E[R_1] + E[D] = (12 - \rho)E[R_1]/8,$$

where $E[R_1] = 1/(\mu - \lambda)$ is the mean of the response-time of a single M/M/1 queue, $\rho = \lambda/\mu$ is its utilization, and D is obtained using Little's law, utilizing the average number of tasks that have completed service and are waiting for their siblings to complete. An approximation is then formulated for a r-node fork-join queue, with $2 \leq r \leq 32$, by observing that the upper and lower bounds of the mean response-time increase at the same rate. The bounds are derived using properties of the random variables concerned: in particular, if the random variables $\{X_1, X_2, \ldots, X_n\}$ are *associated*, i.e., the covariance of $f(\ldots X_i \ldots)$ and $g(\ldots X_i \ldots)$ is positive for any pair of increasing functions f, g,

$$P(\max_{1 \leq i \leq n} X_i > x) = 1 - P(\bigwedge_{i=1}^{n} X_i \leq x) \leq 1 - \prod_{i=1}^{n} P(X_i \leq x).$$

The upper bound is attained when the random variables $\{X_1, X_2, \ldots, X_n\}$ are independent. It can be shown that the response-times R_i are associated and so

$$P(R_r > t) = P(\max_{1 \le i \le r} R_i > t) \le 1 - \prod_{i=1}^{r} P(R_i \le t) = 1 - (1 - \exp^{-(\mu-\lambda)t})^r,$$

the expected value of which is

$$E[R_r] = \int_0^\infty (1 - P(R_r \le t))\mathrm{d}t \le \int_0^\infty (1 - (1 - \exp^{-(\mu-\lambda)t})^r)\mathrm{d}t = \int_0^1 \frac{1-u^r}{(\mu-\lambda)(1-u)}\mathrm{d}u = \frac{H_r}{\mu-\lambda}$$

where H_r is the harmonic series $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{r}$. Similarly, the lower bound H_r/μ is obtained by neglecting queueing effects, so that $H_r/\mu \le E[R_r] \le H_r/(\mu - \lambda)$.

The tightness of the bounds is not the main concern, but the authors conclude that $E[R_r]$ must grow at rate H_r since both bounds grow at that same rate. The mean response-time of a $r > 2$-node fork-join queue can then be expressed as $E[R_r] = S_r(\rho)E[R_2]$, for $r \ge 2$, where the scaling vector $S_r(\rho)$ is approximated by $S_r(\rho) = \alpha(\rho) + (1 - \alpha(\rho))H_r/H_2$ and $\alpha(\rho)$ is to be determined. It follows from simulation results for r =4, 8, 16 and 32 that $\alpha(\rho) \approx 4\rho/11$ provides a good approximation. Thus

$$E[R_r] \approx \left[\frac{H_r}{H_2} + \frac{4\rho}{11}(1 - \frac{H_r}{H_2})\right], \text{ for } r \ge 2. \tag{6.3}$$

Using Eq. (6.3), relative errors of less than 5 percent were obtained for r =4, 8, 16 and 32.

6.5.2 Efficient approximation for tails (EAT)

EAT relies on a similar observation. However, whilst [NT88] approximates the mean response time in homogeneous r-node fork-join queues with exponential inter-arrival and service times for $2 \le r \le 32$, we derive the corresponding approximation for response time *percentiles* in more general r-node fork-join queues with MAP arrivals and PH service times, for unlimited $r > 2$. We start with an approximation for fork-join queues with Poisson arrivals and exponential service times, then extend to queues with MAP arrivals and PH service. We first recap on the

necessary background in order statistics and extreme-value theory.

The maximum random variable is a special case of an *order statistic*, the last one, and it is convenient to express our analysis more generally in terms of order statistics. Let $X_{(1)}, X_{(2)}, \ldots, X_{(r)}$ be the ascending ordered sequence of random variables $X_1, X_2, \ldots, X_r$ (with respect to the normal arithmetic ordering), i.e. $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(r)}$. Then $X_{(i)}$ is the i^{th} order statistic of $X_1, X_2, \ldots, X_r$ $(1 \leq i \leq r)$ [DN81]. In particular, $X_{(r)}$ is the maximum, one of the two *extreme* order statistics (1^{st} and r^{th}). The cumulative distribution function (CDF) of the maximum of r *independent* random variables is

$$F_{X_{(r)}}(x) = \prod_{i=1}^{r} F_i(x),$$

where $F_i(x) = P(X_i \leq x)$. When the X_i are also identically distributed with common CDF $F(\cdot)$, this simplifies to $F_{X_{(r)}}(x) = F^r(x)$.

For i.i.d. random variables with known distribution functions, so-called extreme theory can be applied to approximate the distribution of extreme values, with increasing accuracy as the number of random variables increases. There are three types of distributions for extreme values, among which Type I relates to sets of random variables with exponential tails [Gum54] and includes the *Gumbel distribution* (also called the log-Weibull distribution), with CDF $\exp(-\exp^{-(x-\mu)/\beta})$, for $x \in \mathbb{R}$, which has mode μ (location of the peak of the density function), mean $\mu + \gamma\beta$, and variance $\beta^2\pi^2/6$, where γ is Euler's constant. The *standard Gumbel distribution* has $\mu = 0$ and $\beta = 1$, giving

$$G(x) = \exp(-\exp^{-x}).$$

As $r \to \infty$, the maximum order statistic $X_{(r)}$ for type I distributions is asymptotically distributed as G with normalizing constants a_r and b_r, i.e.,

$$P((X_{(r)} - b_r)/a_r \leq x) = F^r(a_r x + b_r) \to G(x) \text{ as } r \to \infty. \tag{6.4}$$

Here b_r and a_r is the mode and the measure of dispersion of the Gumbel distribution. In the following, we exploit this approximation of extreme values to derive an efficient and accurate approximation for the tail of the response-time distribution in r-node fork-join queues. The approximation requires values for the response-time tail when $r = 2$, for which it uses the model detailed in Section 6.4.

6.5.3 Asymptotic approximation for Poisson arrivals and exponential services

We first consider a r-node fork-join queue where the constituent nodes are M/M/1 queues. We assume the system is homogeneous, i.e. that the M/M/1 queueing nodes are stochastically identical, having the same defining parameters. By using Eq. (6.5), which we derive below, we can replace the time consuming analytical method for fork-join queues with a large number of nodes by a fast alternative that gives a high degree of accuracy, requiring only the analytical results for a single-node and 2-node fork-join queues. We will evaluate the accuracy of this approximation approach in Section 6.6.

Proposition 5. *Following the conjecture of [NT88] regarding growth rates discussed above, the* p^{th} *response-time percentile* $R_r(p)$ *of a* r*-node homogeneous fork-join queue with Poisson arrivals and exponential service times, is given by*

$$R_r(p) = R_1(p) + (R_2(p) - R_1(p)) \ln(r)/\ln(2), \tag{6.5}$$

where $R_1(p)$ *and* $R_2(p)$ *are the* p^{th} *response-time percentiles of single-node and 2-node fork-join queues with the same arrival and service rates, respectively, and* ln *is the natural logarithm.*

Proof. As explained above, the maximum value in a sample of i.i.d. negative exponential random variables approaches the Gumbel distribution as the sample size increases, i.e., when $X_1, X_2, \ldots, X_r \overset{iid}{\sim} \exp(\alpha)$,

$$F_{X_{(r)}}(x) \approx \exp(-r \exp^{-\alpha x})$$

as $r \to \infty$. It follows that the p^{th} percentile of R_r, which satisfies $F_{X_{(r)}}(x) = p$, can be approximated as

$$R_r(p) \approx L(r,p)/\alpha, \tag{6.6}$$

where $L(r,p) = \ln(r) - \ln\ln(1/p)$.

An upper bound $R_r^u(p)$ on the response-time percentiles is obtained by assuming that the r parallel queues are independent, as in [NT88]. Since the response-time distribution of an M/M/1 queue is exponentially distributed with rate $\mu - \lambda$, where λ and μ are respectively the arrival and service rates of the M/M/1 queue (see [HP92] for example), we have $\alpha = \mu - \lambda$ and so

$$R_r^u(p) \approx \frac{L(r,p)}{\mu - \lambda} = \frac{\ln(r)}{\mu - \lambda} - \frac{\ln\ln(1/p)}{\mu - \lambda}.$$

Hence, the upper bound of $R_r(p)$ grows at rate $\mathcal{O}(\ln(r))$. On the other hand, a lower bound $R_r^l(p)$ is obtained by neglecting queueing effects: in this case, the response-time is the maximum of r *i.i.d.* exponentially distributed service times, each with mean $1/\mu$, i.e., we have

$$R^l(r,p) \approx \frac{L(r,p)}{\mu} = \frac{\ln(r)}{\mu} - \frac{\ln\ln(1/p)}{\mu}.$$

Therefore, this lower bound grows at the same rate $\mathcal{O}(\ln(r))$.

We thus observe that both bounds grow at the same asymptotic rate $\mathcal{O}(\ln(r))$ at large r, from which we conclude that $R_r(p)$ itself must be growing at the same rate, as in [NT88]. Clearly we cannot assert that this conjecture holds at every value of $r \geq 1$. However, the average growth rate of $R_r(p)$ over an interval $r \in [1,k]$ must be $L(k,p)$ *asymptotically* as $k \to \infty$; if it were greater, the upper bound would eventually be violated, if it were smaller, the lower bound would be violated. Consequently, knowing the value of $R_1(p)$, we may write

$$R_r(p) \approx \alpha(p)\ln(r)R_1(p) + \beta(p), \text{ for } r \geq 1, \tag{6.7}$$

where $\alpha(p)$ and $\beta(p)$ are both functions of p, and are to be determined. Substituting r by 1 in Eq. (6.7), we find $\beta(p) = R_1(p)$. To obtain $\alpha(p)$, we utilize the matrix analytic model, from

which we obtain $R_1(p)$ for the single-node, and $R_2(p)$ for the 2-node fork-join queues with the same arrival and service processes. Thus

$$R_2(p) = \alpha(p)\ln(2)R_1(p) + R_1(p),$$

from which we obtain

$$\alpha(p) = \frac{R_2(p) - R_1(p)}{\ln(2)R_1(p)}. \tag{6.8}$$

Combining Eqs. (6.8), (6.7) and $\beta(p)$, we obtain Eq. (6.5). □

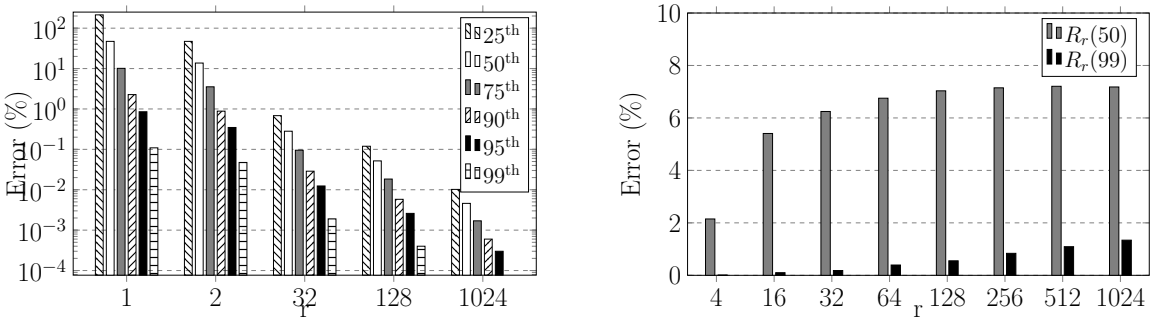
We now provide an example to illustrate this approximation.

Example 1. *We use approximation* (6.5) *to compute the response-time's* 99^{th} *percentile in a 1024-node fork-join queue (*$R_{1024}(99)$*), where the arrival and service rates are* $\lambda = 0.5$ *and* $\mu = 1$*, respectively. To approximate* $R_{1024}(99)$*, we first need the percentiles* $R_1(99)$ *and* $R_2(99)$ *for the same setting. For a single-node queue with* $\lambda = 0.5$ *and* $\mu = 1$*, we obtain* $R_1(99) = 9.215$*. For the 2-node case, we rely on the analytical model presented in Section 6.4 to find* $R_2(99) = 10.555$*, by setting* $C = 100$*. Thus, following* (6.5)*,* $R_{1024}(99)$ *is given by*

$$R_{1024}(99) \approx R_1(99) + (R_2(99) - R_1(99))\ln(r)/\ln(2) = 22.615,$$

while the result from simulation is 22.287, giving a relative error of 1.47%*.*

To further illustrate the approximation, we look at the behavior of the Gumbel distribution as an approximation of the upper bound for this example. Figure 6.2(a) shows the error in the response-time percentiles achieved using the Gumbel distribution, against the actual maximum of the independent exponential random variables that define the upper bound. We depict results for r =1, 2, 32, 128 and 1024, for the 25^{th}*,* 50^{th}*, and* 75^{th} *percentiles, also known as the first three quartiles, and the* 90^{th}*,* 95^{th} *and* 99^{th} *percentiles, representative of the tail. As expected, the response-time distribution of the upper bound converges to the Gumbel distribution with increasing sample size, but the approximation for the tail is significantly more accurate than for the body. For instance, for* $r = 1$*, the errors of the first three quartiles are 213.54%, 47.12%,*



(a) MAP/HE$_2$ (b) $R(50)$, $R(99)$

Figure 6.2: Errors(%) of approximation

and 10.13%, while for the 90th, 95th and 99th percentiles they are 2.27%, 0.85%, and 0.11%, respectively. Since the approximation (6.6) *depends on the values of $R_1(p)$ and $R_2(p)$, we can expect EAT to provide accurate results for the tail of the distribution, even if its accuracy is limited for the first three quartiles. This is also illustrated in Figure 6.2(b), where we show the errors of EAT in approximating the response-time percentiles $R_r(50)$, $R_r(99)$ for values of r between 4 and 1024. Clearly, the error in the 50th percentile is larger than in the 99th percentile, even though both are below 10%, and the latter surpasses 1% error for the largest values of r only. Section 6.6 provides a thorough evaluation of the proposed approximation.*

6.5.4 Asymptotic approximation for MAP arrivals and PH services

We now extend the preceding approximation for exponential service and inter arrival times to queues with MAP arrivals and PH services. The proof in this case requires a more intricate analysis since the extreme value approximations for the upper and lower bounds may have different normalizing constants.

Proposition 6. *The p^{th} response-time percentile $R_r(p)$ of a r-node homogeneous fork-join queue with MAP arrivals and PH services is given by*

$$R_r(p) \approx R_1(p) + (R_2(p) - R_1(p)) \ln(r)/\ln(2), \tag{6.9}$$

given the same growth rate conjecture as in Proposition 5.

Proof. The distribution of the maximum of i.i.d. random variables whose distributions are $\mathrm{PH}(\boldsymbol{\tau}, S)$ converges to the Gumbel extreme-value distribution since they have exponential tails and so are Class I [KS+99, Theorem 9]. However, different PH representations converge to Gumbel distributions with different normalizing constants, where the two main cases considered in [KS+99] are:

- Case 1: if S is irreducible, the distribution of the maximum converges to a Gumbel distribution with normalizing constants

$$a_r = 1/\eta, \quad b_r = \eta^{-1} \ln(Kc), \tag{6.10}$$

  where $-\eta$ is the eigenvalue of S with largest real part, and $c = \boldsymbol{\tau}\boldsymbol{v}$, with $\boldsymbol{v}$ its right eigenvector.

- Case 2: if no two states in S communicate with each other, i.e., if S can be expressed as an upper-diagonal matrix, then the distribution of the maximum converges to the Gumbel distribution with normalizing constants

$$a_r = 1/\eta, \quad b_r = \eta^{-1} \left[\ln(r) + (\upsilon - 1) \ln\ln(r) - \ln(\upsilon - 1)! + \ln(\gamma)\right], \tag{6.11}$$

  where the parameters η, υ, γ are related to the set of paths from any initial state to the absorbing state, as described in [KS+99, Remark 10].

In the following we show that both upper and lower bounds of the response-time percentile $R_r(p)$ grow at a rate $\mathcal{O}(\ln(r))$ if the PH service-time distribution falls into either of these two cases.

As above, the response-time percentiles in the case that the parallel queues are independent provide upper bounds for the r-node fork-join queue. Each of these queues is then a MAP/PH/1 queue, the response-time of which follows a PH distribution with parameters $(\boldsymbol{\alpha}^1_{\mathrm{res}}, S^1_{\mathrm{res}})$, from which the eigenvalue and eigenvector pair $(\eta^u, \boldsymbol{v}^u)$ is obtained. To compute $\mathrm{PH}(\boldsymbol{\alpha}^1_{\mathrm{res}}, S^1_{\mathrm{res}})$, we rely on the Q-MAM tool [PVVVH08], which utilizes the age-process approach originally

proposed in [Sen89]. Moreover, from [Sen89, Lemma 2.5], we know that the matrix S^1_{res} is irreducible as long as the queue is stable. Hence we can use Eq. (6.4) and the normalizing constants in Eq. (6.10) to obtain an approximation for the maximum $X_{(r)}$ of a set of r independent PH variables. It follows that

$$F_{X_{(r)}}(x) \approx \exp\left(-\exp^{-(x-b_r)/a_r}\right) = \exp\left(-\exp^{-(\eta^u x - \ln(rc^u))}\right) = p,$$

where $c^u = \boldsymbol{\alpha}^1_{\text{res}} \boldsymbol{v}^u$, so that the upper bound of $R^u_r(p)$ has the form

$$R^u_r(p) \approx \frac{\ln(r)}{\eta^u} + \frac{\ln(c^u) - \ln\ln(1/p)}{\eta^u}.$$

Therefore the upper bound of $R_r(p)$ grows at asymptotic rate $\mathcal{O}(\ln(r))$.

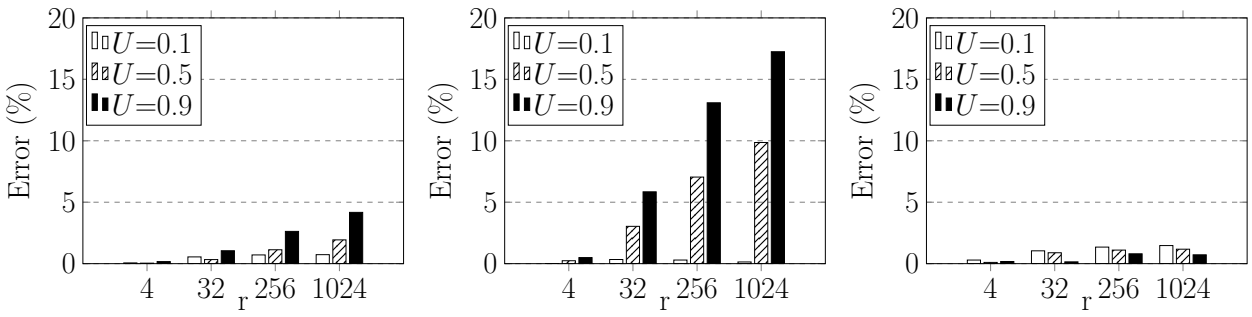
For the lower bound, we again ignore queueing delays, thus taking the maximum of the service time in the r queues, each of which follows a PH distribution with parameters $(\boldsymbol{\alpha}_{\text{task}}, S_{\text{task}})$. Different from the upper bound, where the matrix S^1_{res} is always irreducible as long as the queue is stable, the matrix S_{task} may fall into either of the 2 cases mentioned. Below we show that, in either of these two cases, the lower bound of $R_r(p)$ grows at a rate $\mathcal{O}(\ln(r))$. In the case that S_{task} is irreducible, let $-\eta^l$ be the eigenvalue with largest real part of S_{task}, and $\boldsymbol{v}^l$ its associated right eigenvector. Then the lower bound of $R^u_r(p)$ has the form

$$R^l_r(p) \approx \frac{\ln(r)}{\eta^l} + \frac{\ln(c^l) - \ln\ln(1/p)}{\eta^l},$$

which grows at the same rate $\mathcal{O}(\ln(r))$. If S_{task} falls into case 2, e.g., the S_{task} corresponds to an Erlang or hyper-exponential distribution, the normalizing constants are given by Eq. (6.11), and it follows that the lower bound $R^l_r(p)$ has the form

$$R^l_r(p) \approx \frac{[\ln(r) + (\upsilon - 1)\ln\ln(r)]}{\eta^l} + \frac{[-\ln(\upsilon - 1)! + \ln(\gamma) - \ln\ln(1/p)]}{\eta^l}.$$

Thus the lower bound of $R_r(p)$ grows asymptotically at rate $\mathcal{O}(\ln(r))$, as the term $\ln\ln(r)$ is dominated by $\ln(r)$ [Sip06]. Consequently, we conclude that both bounds grow at the same rate



(a) Exp/Exp (b) MAP/Exp (c) Exp/HE_2

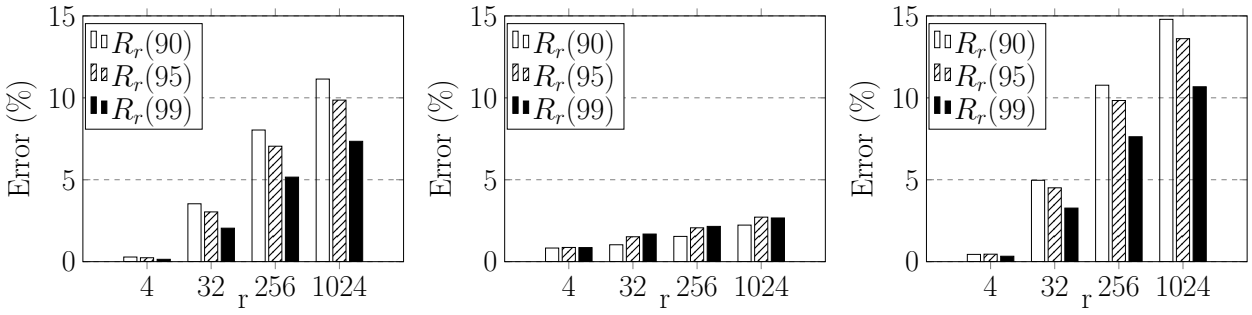
Figure 6.3: Errors(%) of approximation for $R_r(95)$

$\mathcal{O}(\ln(r))$. Following a similar procedure to that for queues with Poisson arrivals and exponential services, we obtain Eq. (6.9). □

6.6 Evaluating the EAT approximation

To evaluate the quality of the EAT approximation, we compare its results against simulation for a broad range of system setups. For each setting, the response time percentiles and their 95% confidence intervals were obtained after 5,000 simulations with 200,000 samples each time. We considered different utilization levels, service time distributions, and arrival processes, following the same parametrisation as in Section 6.4.5. To obtain the values for the $r = 2$ case, on which EAT relies, we used $C = 500$ and 1000 to ensure accurate results.

Figure 6.3 shows the relative error of $R_r(95)$ in cases with number of nodes $r = 4, 32, 256, 1024$ and at load levels of 0.1, 0.5 and 0.9. Figure 6.3(a) considers exponential service times and Poisson arrivals, where we observe that EAT produces accurate results, the error being as low as 4.17% for the most challenging case with $r = 1024$ at load 0.9. We also observe that the error increases with the load, although the errors remain below 5%. Replacing the arrival process by a MAP, as depicted in Figure 6.3(b), we observe an increase in EAT's errors, particularly at high utilization. Instead, replacing exponential service times by hyper-exponential, Figure 6.3(c) shows how the errors are actually smaller, staying under 2% even at large r. This result suggests that EAT performs better with more variable service times, but worse with more variable inter-arrival times.



(a) MAP/Exp (b) MAP/HE_2 (c) MAP/ER_2

Figure 6.4: Errors(%) of approximation when $U = 0.5$

Since MAP arrivals are the most challenging, Figure 6.4 focuses on this case, fixing the utilization at 0.5 and considering different response-time percentiles and service time distributions. Figure 6.4(a) shows how the error increases with r, but decreases with the percentile. In fact, the lowest errors are achieved for the 99^{th} percentile, a result that is related to the faster convergence of the Gumbel distribution for higher percentiles, as discussed in Section 6.5. If we replace the service-time distribution by a hyper-exponential, we observe a large reduction in the errors, which now remain under 3%. The opposite effect occurs if Erlang service times are considered, as the errors increase to as much as 15%. We therefore observe that EAT offers much better results under highly-variable service times, even under MAP arrivals. Low variability in the service times appears more challenging for EAT, although the errors are remarkably small considering that we approximate cases as large as $r = 1024$ with data from the cases $r = 1$ and $r = 2$ only.

6.7 Discussion

We have proposed a stochastic model for r-node, homogeneous fork-join queues with phase-type processing times and MAP arrivals. Instead of focusing only on the mean response-time, this model facilitates the estimation of the percentiles of response-time – its distribution – and offers accurate results for small-scale fork-join queues. For large-scale fork-join queues, where the numerics of the stochastic model become unviable, we proposed an efficient and scalable approximate approach, which only requires the analytical results for a single-node and 2-node fork-join queues. Tests against simulation show that the approximation gives a high degree of

accuracy for the response-time tails, so that, since mean values can be approximated by known methods, these models cover the majority of response-time requirements in such queueing systems. Future work may consider the extension of the model to consider heterogeneous servers. Also, the efficiency of the approximation makes it a good candidate to support resource provisioning decisions in parallel computing frameworks.

Chapter 7

Managing Response-Time Tails by Erasure Encoding

Storage costs are increasingly taking a higher proportion of corporate operating and capital budgets as data growth continues unabated. As a result, ways are being sought to reduce costs by simplifying and consolidating operations while maintaining service level objectives (SLOs) for performance, data management and data protection. Scale-out systems use commodity servers as nodes have become increasingly popular for both on-premise and cloud storage. As an example of one type of architecture, consider a cluster of nodes where each node maintains its own set of persistent storage devices (such as Solid State Drives, or SSDs) and some form of erasure encoding provides protection from one or more node failures. With erasure encoding, an object is stored by creating k fragments and storing $r \geq k$ encoded fragments so that the original object can be retrieved from any k of the r fragments (whence $r - k$ failures can be tolerated). Besides the failure-protection, many architectures also harvest response time (i.e. data access time) benefits, for example by issuing r tasks (corresponding to fragments) and constructing the required object after the first k responses have been received. These can be described as (r, k) *partial-join queues* where the special case with $k = 1$ is simple replication [CSH+14, SLR12, HPZR12, JLS14] and $r = k + 1, r = k + 2$ or $r = k + 3$ can represent single, dual or triple parity scenarios.

Partial-join queues can be implemented with various policies based on when redundant tasks are canceled, for example: (a) "*Canceling*", in which all associated tasks terminate their service immediately after the first k complete; (b) "*Early canceling*", in which $r - k$ associated tasks leave the system when any k of the tasks have *started* service. As a benchmark, we also consider a baseline model of a *split-(partial-)merge* queue, in which objects are processed FCFS and a service time is the time for the first k out of r tasks of an object to complete, whereupon the "merge" occurs. Thus, accesses are synchronous in that only one "split" can be pending at any time, so that servers will be idle when some of them (fewer than k) have completed.

This chapter obtains stochastic models, incorporating controlled approximations, for response times in systems operating the aforementioned three policies. The key contributions are as follows:

- Response-time distribution functions are computed, rather than just its moments, like the average value, as considered by, for example, [CSH$^+$14, SLR12, HPZR12, JLS14, XLAC14, LK14].
- In contrast to previous results that consider a single queue at a centralized dispatcher [LK14, CSH$^+$14, HPZR12, XLAC14], we investigate systems with an individual queue in front of each of the servers, which process tasks asynchronously but are not independent as every new arrival – from a MAP – sends one task to every queue simultaneously.
- Extensive experimentation with the models enable the comparison of the policies considered. For instance, we show that assuming no canceling overhead canceling always performs best, particularly at low utilizations. However, early canceling may actually be preferable at higher utilizations if canceling has a high overhead or is harder to implement.
- As a result, sound recommendations as to redundancy policies can be made for capacity planning in distributed storage systems, choice of redundancy level and fine-tuning of system parameters, such as the granularity of fragments.

The key issue addressed by the present work is to deal with *general order statistics* of task response times, rather than just the special cases of the maximum and minimum, correspond-

ing to first-to-finish replication in Chapter 5 and fork-join queue in Chapter 6, respectively. Even with split-merge policies, where all a job's tasks are allocated to servers simultaneously immediately after the completion of service of the previous job, the problem is not straightforward; see Section 7.5. More realistic problems, with proper partial-joins, are investigated using matrix-analytic methods. The partial-join operation must be modeled differently here since a simple order statistic could not account for the presence of tasks from other jobs in the queues, which break the synchronicity of split-merge. The only way to account for partial-joins accurately is to keep track of the states of all the queues as well as the progress of the particular tasks of some marked job (of which the response time is sought), which results in a huge raw state space. Here we follow the idea in Section 3.3, by limiting the queue-length difference to be at most an integer $C < \infty$. This renders the matrix-analytic approach tractable.

7.1 Related work

Several recent works have studied the effectiveness of coding on latency reduction in (r, k) partial-join queues [CSH$^+$14,SLR12,HPZR12,JLS14,XLAC14,LK14]. In one variant [XLAC14, HPZR12] each job creates k tasks, which can be served by any k out of the r servers, and departs only after all its tasks complete service. Models that generate r tasks per job and wait for any k of them to complete service, with various policies for handling the remaining $r-k$ tasks, are considered by [LK14, CSH$^+$14, JLS14, SLR12]. In particular, [SLR12] argues that, under certain conditions, the mean response time is minimized when tasks are sent to all servers. Among these works, [LK14, CSH$^+$14, HPZR12, XLAC14] consider a centralized set-up, where a central queue buffers jobs when all the servers are busy. On the other hand, [JLS14] derives upper and lower bounds on the mean response time for a distributed set-up, where upon the arrival of a job, each task is dispatched immediately to a different server, each of which maintains a local queue. This approach simplifies the dispatching logic and is often used in server farms at scale. We have also considered the distributed set-up in which all tasks are sent to all r servers on a job arrival. The early-canceling policy has also been considered for replication to reduce latency, as in [JSW15, DB13], for example. However existing studies obtain the *mean* response

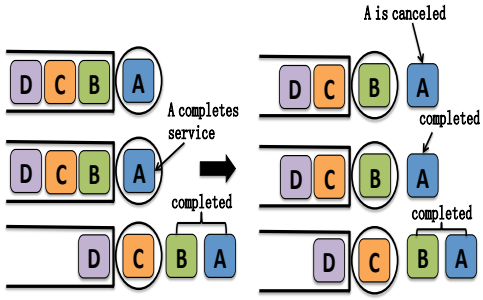


Figure 7.1: Canceling example

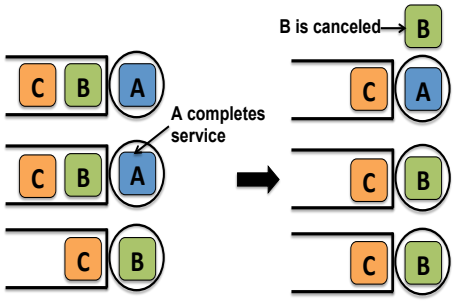


Figure 7.2: Early-canceling example

time [CSH+14, SLR12, HPZR12, JLS14, XLAC14, LK14], whereas our work compares policies in the context of response-time distribution.

7.2 Model types and definitions

The model we consider comprises r parallel servers, each with its own FCFS queue, which serves requests with content. Requests first arrive at a scheduler, where they are transformed into jobs (e.g., representing objects in a storage system) composed of r tasks (representing fragments, for example). Each of these tasks is then submitted to a different server, where the scheduler keeps track of their service completion status. Thanks to the coding scheme, it is only necessary to wait for $k < r$ tasks to complete, as these carry sufficient information to reconstruct the entire content requested. The response time of a job is therefore the time interval between its arrival and the time its k^{th} task completes service. The question is, what is to be done with the remaining $r - k$ tasks? In the following section we consider two scheduling policies for these $r - k$ tasks.

7.2.1 Scheduling policies

1. Canceling: Immediately after the first k tasks of a job complete service, the remaining $r - k$ tasks are canceled, i.e., leave their queue (or server, if currently being served). This policy avoids the extra load that could be introduced by the redundant tasks, leading to better response time. On the downside, the checking associated with the erasure code has to be abandoned, although this is not a problem in many storage systems that rely on other data

protection mechanisms, such as block checksums. To illustrate the operation of this policy, Figure 7.1 depicts an example with $r = 3$ servers, where the job is done when any $k = 2$ out of 3 tasks are finished. When the task of job A in the middle server completes service, the whole job A also completes service since one task had already finished in the lower server. The outstanding task in the upper server is therefore canceled. Both of the tasks of job B that are waiting in the queues start service.

2. Early-canceling: As soon as the first k tasks of a job have *started* service, the remaining $r-k$ tasks are canceled immediately. This policy is appropriate in applications where the latency is mainly due to queuing delays [DB13]; in many systems, once a request is in execution, the variability of its remaining response time decreases substantially. We illustrate this policy in Figure 7.2, again for the case with $r = 3$ and $k = 2$. Here, we start from a configuration where all the queues are empty and job A arrives. Arbitrarily, two of its tasks start service in the upper and middle servers. Its third task is canceled, allowing a task of job B to enter service in the lower server before either of the tasks of job A have finished in the upper or middle servers. Job C also arrives before any service completions and we obtain the configuration on the left in the figure. When task A completes service in the middle server, task B in the middle queue starts service. As job B now has two tasks that have started service, its last one in the top (longest) queue is canceled.

3. Split-merge model: In this model, all tasks of a job start service at the same time. $k-1$ servers remain idle after completing their tasks until the completion of the k^{th} task, whereupon the job completes service and leaves. Thus, this model implements canceling but with resource starvation. This model is simple to analyze and provides an upper bound on the response time for other models. Section 7.5 extends this model to MAP arrivals and phase-type service times.

7.3 Canceling model

In the canceling case, upon completion of the first k tasks of a job, the remaining $r - k$ are canceled immediately. In the special case that $k = 1$, the synchronization introduced by the

canceling mechanism simplifies the system to a $MAP/M/1$ queue with service rate $r\mu$ as all r tasks execute simultaneously and the first one to complete cancels all other tasks, allowing a new job to start service with all its r tasks. Also, when $k = r$, the system becomes a traditional fork-join queue as we have to wait for all the tasks to complete service. The main modeling issues arise from the fact that the system is a set of parallel queues, the evolution of which is tightly coordinated as every arriving job sends a task to each queue. Furthermore, task canceling can occur when the redundant tasks are either waiting in the queue or in service, which requires accurately tracking the queues' states. In addition, a job's service time depends on the complete system state, a feature we need to account for when describing response times. In the general case with $1 \leq k \leq r$, we show that both the job waiting time and job service time have PH distributions, and that these can be combined to yield the required job response-time distribution.

7.3.1 The waiting-time distribution

To determine the waiting-time distribution, we perform an age-based analysis [AM01], observing the r queues only during the *all-busy* periods, i.e., when the r servers are all busy, and defining a bivariate Markov process $\{X(t), J(t)|t \geq 0\}$. The *age* $X(t)$ keeps track of the total time-in-system of the current youngest job in service, and *phase* $J(t) = (A(t), S(t))$ holds the joint state of the MAP arrival process $A(t)$ and the service process $S(t)$. The service process keeps track of the r queue lengths, where the length of a queue includes both the tasks in service and waiting in the queue. Before fully defining $S(t)$ we note the following property.

Proposition 7. *In the system with canceling, the longest $r - k + 1$ queues always have the same length.*

Proof. The oldest job in service cannot have any older tasks in front of any of its tasks in any queue. In other words, in every queue the oldest job in service must either be in-service or else have already completed service. Moreover, every queue at which this job has a task in-service

must have the same length, since each of them has exactly one task waiting to commence service emanating from each of the jobs that arrived after the oldest job in service. Clearly, these are the longest queues. To complete the proof, we note that the maximum number of queues from which a task of the oldest job has departed is $k-1$, for otherwise the job would have finished, with all its remaining tasks canceled. Hence the minimum number in service is $r-k+1$. □

This effect is illustrated in Figure 7.1 where, with $r=3$, $k=2$, the 2 longest queues decrease their length at the same time when either task of job A completes.

Due to this proposition, instead of keeping track of the states of all queues, we focus on the shortest k queues only. Furthermore, instead of considering the k queue-lengths themselves, we order them by length to obtain the queue-length difference between any queue and the shortest one. And, since all the queues are homogeneous, it suffices to count the number of queues with the same difference. Thus, the service process $S(t)$ keeps track of the number of queues with i more tasks than the shortest queue, $n_i(t)$, for $i \geq 0$. We follow Section 3.3 to truncate the state space artificially by limiting the difference to be at most some positive integer $C < \infty$, so that $S(t)$ is given by $(n_0(t), n_1(t), \ldots, n_C(t))$, which takes values in the set

$$\Big\{(n_0, \ldots, n_C) \mid \sum_{i=0}^{C} n_i = k, n_0 \geq 1, n_i \geq 0 \ (0 < i \leq C)\Big\},$$

where n_0 is the number of shortest queues. For instance, assume $C=2$ and consider the scenario on the left in Figure 7.1 with $r=3$ and $k=2$. There, the service process $S(t)$ is in state $(n_0(t), n_1(t), n_2(t)) = (1,0,1)$, as there is one shortest queue, and one queue representing the longest $k=2$ queues, both of which have two more tasks than the shortest one. Under this approximation, the number of states is

$$m_s = \binom{k+C}{k} - \binom{k+C-1}{k} = \binom{k+C-1}{C},$$

so that the phase process $J(t)$ takes values in a set of size $m = m_a m_s$, where m_a is the number of arrival phases, as defined in Section 7.2.

Table 7.1: Transition rates for matrices S and $A^{(\text{jump})}$

From	To	Rate	Matrix
$(n_0, n_1, \ldots, n_{C-1}, n_C)$	$(1, n_0 - 1, n_1, \ldots, n_{C-2}, n_{C-1} + n_C)$	$n_0\mu$	$A^{(\text{jump})}$
$(n_0, \ldots, n_{\text{longest}}, 0, \ldots, 0)$	$(n_0, \ldots, n_{\text{longest}-1} + 1, n_{\text{longest}} - 1, 0, \ldots, 0)$	$(n_{\text{longest}} + r - k)\mu$	S
$(n_0, \ldots, n_{i-1}, n_i, \ldots, n_C)$	$(n_0, \ldots, n_{i-1} + 1, n_i - 1, \ldots, n_C)$	$n_i\mu$, $1 \leq i <$ longest	

To determine the PH representation of the waiting-time distribution, the next step is to compute the stationary distribution $\boldsymbol{\pi}(x)$ of the process $(X(t), J(t))$, which involves to find the matrix T and vector $\boldsymbol{\pi}(0)$. To determine the $m \times m$ matrix T, we need to find the matrices S and $A^{(\text{jump})}$, we thus summarize in Table 7.1 their transition rates. The matrix $A^{(\text{jump})}$ holds the rates of service completion at any of the shortest queues, as a new job starts service if and only if there is a task service completion in a shortest queue. This is illustrated in Figure 7.1, where a hypothetical job D, arriving after job C, would start service if the task of job C in the lower server (shortest queue) completes service. Alternately, other task service completions would allow a task of job B to start service, but job B must have started service previously as one of its tasks had already left the lower server. The first row of Table 7.1 shows that a service completion in any of the shortest n_0 queues, results in a single shortest queue, together with $n_0 - 1$ queues with one more tasks in queue than the shortest one. It also shows a possible subtle effect of the truncation.

Since we observe only the shortest k queues, assuming n_{longest} queues of these are the longest, there must be $n_{\text{longest}} + r - k$ queues among all the r queues with this length, as the longest $r - k + 1$ queues always have the same length. This is captured in matrix S, as shown in the second row of Table 7.1, where a task completes service in one of these longest queues with transition rate $(n_{\text{longest}} + r - k)\mu$. The third row in this table considers service completions in queues other than the shortest and longest ones, where a task in service in one of the queues with difference n_i completes service, reducing its difference with respect to the shortest queue by one, for $i \geq 1$. Notice that as the queue-length differences are bounded by C, transitions leading these differences to exceed the limit C are re-assigned to n_C.

To find the steady-state distribution $\boldsymbol{\pi}(0)$ of the phase at the beginning of an all-busy period, we keep track of the system evolution during the not-all-busy period and the transitions connecting between the not-all-busy and all-busy periods. During a not-all-busy period, the shortest queues

Table 7.2: Transition rates for $S_{\text{(full)-(not-full)}}$ and $S_{\text{not-full}}$ (Canceling)

From	To	Rate	Matrix
$r, (n_0, \ldots, n_i, \ldots, n_C)$	$r-1, (1, n_0-1, \ldots, n_{C-2}, n_{C-1}+n_C)$	$n_0\mu$	$S_{\text{(full)-(not-full)}}$
$r, (n_0, \ldots, n_{\text{longest}}, 0, \ldots, 0)$	$r, (n_0, \ldots, n_{\text{longest}-1}+1, n_{\text{longest}}-1, 0, \ldots, 0)$	$(n_{\text{longest}}+r-k)\mu$	$S_{\text{not-full}}$
$r, (n_0, \ldots, n_{i-1}, n_i, \ldots, n_C)$	$r, (n_0, \ldots, n_{i-1}+1, n_i-1, \ldots, n_C)$	$n_i\mu$, $1 \le i < \text{longest}$	
$r, (n_0, n_1, \ldots, n_C)$	$r-1, (n_0+1, n_1-1, \ldots, n_C)$	$n_1\mu$, $r > r-k+1$	

must be empty and the system will remain in the not-all-busy period until a new arrival occurs, sending tasks to all queues and initiating an all-busy period. Thus, the system's evolution is governed by service completions and it suffices to keep track of the phase process $J(t)$, which is defined similarly as for the all-busy period. To hold the transition rates across service phases, between arrivals, during a not-all-busy period, we introduce the $m_s \times m_s$ matrix $S_{\text{not-all-busy}}$, whose non-negative transition rates are actually the same as in matrix S since this matrix also contains service completion rates between arrivals. Since the shortest queues are empty, they have no service completions. Finally, we set the diagonal entries of matrix $S_{\text{not-all-busy}}$ to make it a Markov chain generator. Then the vector $\boldsymbol{\pi}(0)$ can be found by solving Eq. (3.13).

After $\boldsymbol{\pi}(0)$ has been obtained, we follow the steps in [AM01] to find the PH representation of the waiting-time distribution.

7.3.2 The service-time distribution

The job service time under the canceling scheme is the time elapsed between the start of service of its first task and the time when its k^{th} task completes. We thus follow a tagged job and let $Y(t) = (R(t), S(t))$ be the service state of this job at time t. Here $R(t)$ records the number of tasks in the tagged job that *have not finished service* at time t (recall that $S(t)$ is the state-vector of the queues). Since the tagged job completes service when any k of its r tasks complete service, we are interested in service phases where at least $r-k+1$ tagged tasks are still in service or queuing, i.e., when $R(t) \ge r-k+1$. Although similar to the service phase defined for the waiting-time distribution, in this case the variable $S(t)$ focuses on the queue-lengths *in front of the tagged job only*. Thus, if $R(t) = r$, n_0 holds the number of queues with a tagged task in service, and $\sum_{i=0}^{C} n_i(t) = R(t)$. Instead, if $R(t) < r$, n_0 holds the number of servers that have already finished serving tagged tasks, and $\sum_{i=1}^{C} n_i(t) = R(t)$. Note that when $R(t) < r$,

the number of tagged tasks currently in service is given by n_1. Assuming a bound on the queue-length difference $C = 2$ and considering the example on the left in Figure 7.1, if job C is the tagged job, then $n_0 = 1$ and $S(t) = (1, 0, 1)$, as there is one queue among the shortest $k = 2$ queues that has two more tasks than the shortest one. If instead we focus on job B, then $S(t) = (1, 1, 0)$.

Next, to obtain the service-time PH representation, we split the service states into two sets: the *full* phases, where none of the tagged tasks have completed service, so that $R(t) = r$; and the *not-full* phases, where at least one tagged task has already finished service, so that $r - k + 1 \leq R(t) < r$. We then put together the transition rates among service states in a sub-generator S_{ser}, which is partitioned according to the above two subsets as:

$$S_{\text{ser}} = \begin{bmatrix} S_{\text{full}} & S_{\text{(full)-(not-full)}} \\ 0 & S_{\text{not-full}} \end{bmatrix} \otimes I_{m_a}. \tag{7.1}$$

Here the matrix S_{full} holds the rates of transitions that do not involve a service completion in the shortest queues while all the servers are busy; thus it is the same as the matrix S. When the tagged job is in a full state and a service completion in the shortest queues occurs, the tagged job jumps to a not-full state, as shown in the first row of Table 7.2. Once the job is in a not-full state, the matrix $S_{\text{not-full}}$ considers three types of transitions, summarized in rows 2-4 of Table 7.2: the second row considers the case where a task completes service in one of the longest queues; the third row covers the case with service completions at any queue but the longest and shortest ones; and in the fourth row, a tagged task completes service.

To complete the PH representation of the service-time distribution, the next step is to find the stationary distribution of the phase $\boldsymbol{s}_{\text{ser}}$ in which a task starts service, for which we rely on Proposition 3. Having obtained the PH representations of a job's waiting and service times, the PH representation of the response-time distribution follows from Eq. (6.2).

7.3.3 Resource consumption

The resource consumption, or system utilization, is the expected fraction of time that a server is busy. Under the canceling scheme, assuming $E[r]$ is the expected number of servers used by a job, the utilization can be computed as $\rho = E[r]\lambda/(r\mu_{\text{job}})$, where λ^{-1} is the mean job inter-arrival time, and μ_{job}^{-1} is the mean job service time, given by $\mu_{\text{job}}^{-1} = \boldsymbol{s}_{\text{ser}}(-S_{\text{ser}})^{-1}\mathbf{1}$. As requests may finish service using k or more servers depending on their evolution amongst service phases, thus $k \leq E[r] \leq r$, and $E[r]$ can be obtained as $E[r] = \langle \boldsymbol{\pi}_{\text{ser}}, \boldsymbol{n}_{\text{server}} \rangle$, where $\langle \cdot, \cdot \rangle$ is the inner-product operator. Here $\boldsymbol{\pi}_{\text{ser}}$ is the stationary distribution of the job service phase, given by $\boldsymbol{\pi}_{\text{ser}} = \boldsymbol{s}_{\text{ser}}(-S_{\text{ser}}^{-1})/(\boldsymbol{s}_{\text{ser}}(-S_{\text{ser}})^{-1}\mathbf{1})$. Also, the i^{th} element of the vector $\boldsymbol{n}_{\text{server}}$ holds the number of tagged tasks in service in the i^{th} service phase, which is equal to r for full states, and to n_1 for not-full states.

7.4 Early-canceling model

Under the early-canceling policy, described in Section 7.2, the redundant $r - k$ tasks are canceled once the first k start service. We follow a similar analysis as in the previous section, determining first the waiting-time distribution and then the service-time distribution. However, the structure of the state-space and the state-transitions differ significantly. Under the early-canceling policy, the system simplifies to a MAP$/M/r$ queue when $k = 1$, as one task of each job starts service, canceling all its siblings, as long as there is a server available; and, if all servers are busy, the tasks all wait in a FCFS queue. Instead, when $k = r$, the system is equivalent to a traditional fork-join queue, as with canceling. To analyze the general case with $1 \leq k \leq r$, we note the following property.

Proposition 8. *In the system with early canceling, the longest $r - k + 1$ queues always have the same length.*

Proof. The proof is inductive and in four parts.

Table 7.3: Transition rates for matrices $S_{\text{(all),(not-all)}}$ and $S_{\text{not-all-busy}}$ (early-canceling)

From	To	Rate	Matrix
$(n_0, n_1, \ldots, n_{C-1}, n_C)$	$(1, n_0 - 1, n_1, \ldots, n_{C-2}, n_{C-1} + n_C)$	$n_0\mu$	$S_{\text{(all),(not-all)}}$
$(n_0, \ldots, n_{\text{longest}}, 0, \ldots, 0)$	$(n_0, \ldots, n_{\text{longest}-1} + n_{\text{longest}}, 0, \ldots, 0)$	$n_{\text{longest}}\mu$	$S_{\text{not-all-busy}}$
$(n_0, \ldots, n_{i-1}, n_i, \ldots, n_C)$	$(n_0, \ldots, n_{i-1} + 1, n_i - 1, \ldots, n_C)$	$n_i\mu$, $1 \leq i < \text{longest}$	

(1) In the state where all the queues are empty and the servers idle, the property holds vacuously, and we assume inductively that it holds at all times.

(2) In any state of the queues, a new arrival has up to k of its tasks start service (by arriving at an idle server), at least $r - k + 1$ tasks join the longest queues, or no tasks join a queue if k tasks started service and $r - k$ were canceled. In either case, the longest $r - k + 1$ queues remain at the same length.

(3) After a service completion, if the k^{th} task of a job enters service, the queue length at that server decreases by one and the task's siblings are canceled. Hence the $r - k + 1$ longest queues all lose one task and remain at the same length.

(4) If after a service completion a task of a job that is not its k^{th} enters service, it could not have come from one of the $r - k + 1$ longest queues. Its queue length therefore reduces by one and the $r - k + 1$ longest queues remain the longest and unchanged in length.

As a result, the $r - k + 1$ longest queues remain at the same length at all times. □

This is illustrated in Figure 7.2 where, with $r = 3$, $k = 2$, the two longest queues decrease their length at the same time when the second task of the youngest job in service (B) starts to be served. Due to this proposition, we again need not keep track of the states of all queues, but instead focus on the shortest k queues only – just as with the canceling policy.

7.4.1 The waiting-time distribution

Based on the observation above we focus on the k shortest queues and proceed with an age-based analysis, as in the previous section, defining the age $X(t)$ and the phase $J(t)$, the latter evolving on the same set. Again, we define matrices S and $A^{(\text{jump})}$ and determine the matrix

Table 7.4: Transition probabilities for $R_{\text{all-busy}}$ and $R_{\text{not-all-busy}}$ (early-canceling)

From	To	Probability	Matrix
$(n_0, n_1, 0, \ldots, 0)$	$(n_0 - k, n_1 + k, 0, \ldots, 0)$	1, $n_0 > k$	$R_{\text{not-all-busy}}$
$(n_0, \ldots, n_i, \ldots, n_C)$	$(n_0, \ldots, n_i, \ldots, n_C)$	1, $n_0 \leq k$	$R_{\text{all-busy}}$

Table 7.5: Transition rates for $S_{\text{(full)-(not-full)}}$ (early-canceling)

From	To	Rate
$(n_0, \ldots, n_i, \ldots, n_C)$	$k-1, (1, n_0 - 1, \ldots, n_{C-2}, n_{C-1} + n_C)$	$n_0\mu$

T from Eq. (2.11). Under both schemes, during the all-busy period, a service completion at the shortest queue triggers a new job (if present) to start service, so it is clear that the matrix $A^{(\text{jump})}$ is the same, as defined in Table 7.1. Further, as highlighted in Proposition 8, the $r - k + 1$ longest queues are formed on those queues where the youngest job in service still has tasks waiting. Therefore, if a task in any of these queues completes service, it will trigger all the longest queues to decrease at the same time, as described in the second row of Table 7.1. Finally, service completions in other queues simply allow a new task to start service, modifying the queue-length differences as in the third row of Table 7.1. Thus the matrix S is also the same for both canceling schemes.

Different from the canceling case, in the early-canceling case, more than one arrival may be necessary to initiate an all-busy period. For instance, consider an empty system with $r = 4$ queues and $k = 3$, so that an arrival leads to 3 out of the 4 servers becoming busy, while one server remains empty due to the early canceling, leaving the system in the not-all-busy state. This contrasts with the canceling case, where any arrival during a not-all-busy period must initiate an all-busy period. Whilst during all-busy periods we keep track of the states of the shortest k queues only, during a not-all-busy period we must also consider the number of idle servers and the state of all the non-empty queues. The state-vector is therefore $S^e(t) = \{n_0(t), \ldots, n_C(t)\}$, where n_0 is the number of idle servers at time t. This process takes values on the set $\{(n_0, \ldots, n_C) \mid \sum_{i=0}^{C} n_i = r, n_i \geq 0, n_0 \geq 1\}$, which has cardinality $m_{\text{not-all-busy}}$.

We now define the matrix $S_{\text{(all),(not-all)}}$ to hold the service transition rates that trigger the start of a not-all-busy period from an all-busy-period, i.e., connecting the process $S(t)$ with the process $S^e(t)$ and having size $m_s \times m_{\text{not-all-busy}}$. Table 7.3 shows that the non-zero entries of $S_{\text{(all),(not-all)}}$

hold the service completion rates at the shortest queues, as the all-busy period terminates if any of these queues completes service before the next arrival. Next, we focus on the not-all-busy period and introduce an $m_{\text{not-all-busy}} \times m_{\text{not-all-busy}}$ matrix $S_{\text{not-all-busy}}$, which holds the service transition rates between arrivals during the not-all-busy period, as summarized in Table 7.3. The second and third rows of the table consider a service completion in any of the longest queues, and in any of the other queues, respectively.

Since, during the not-all-busy period, several arrivals could occur, we track service phase changes due to arrivals by introducing an $m_{\text{not-all-busy}} \times m_{\text{not-all-busy}}$ matrix $R_{\text{not-all-busy}}$, the $(i,j)^{\text{th}}$ entry of which is the probability that the service phase just after an arrival is j, given that it was i just before. As shown in Table 7.4, these transitions are restricted to cases where a job finds more than k idle servers upon arrival, thus starting service with all its k tasks, and the system stays in the not-all-busy period. To consider the complementary case where an arrival sees at most k idle servers and initiates an all-busy period, we introduce an $m_{\text{not-all-busy}} \times m_{\text{all-busy}}$ matrix $R_{\text{all-busy}}$, whose non-zero entries are summarized in Table 7.4. The only difference is that the phase after the arrival relates to the process $S(t)$, since the all-busy period has started.

With the definitions above we can find $\boldsymbol{\pi}(0)$, which is the steady state distribution of the phase at the beginning of an all-busy period, as the solution to Eq. (5.5). We can then follow the steps in Section 5.2.2 to find $\boldsymbol{\pi}(0)$. Having found T and $\boldsymbol{\pi}(0)$, we utilize [AM01] to find the PH representation of the waiting-time distribution.

7.4.2 The service-time distribution

We now seek a PH representation $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ for the job service-time distribution. With early-canceling, a job can have at most k tasks in service, as the remaining $r-k$ are canceled when the k^{th} task starts service. Hence the state of a tagged job in service at time t is defined as $Y(t) = (R(t), S(t))$, where $R(t) \in \{k, k-1, \ldots, 1\}$ is now the number of tasks that *have not completed service* and $S(t)$ keeps track of the queue-lengths in front of the tagged tasks. Similar to the canceling case, we identify two subsets for the state space of this process, the full set and the not-full set. The difference here is that a job in service has at most k tasks, thus $R(t) = k$

and $R(t) < k$ for phases in the full and not-full sets, respectively. We can therefore describe the service transition rates in a sub-generator matrix S_{ser}, which we partition as in Eq. (7.1). As the matrix S_{full} holds the rates of service-transitions that exclude service completions of the tagged tasks, it is equal to the matrix S.

The matrix $S_{\text{not-full}}$, on the other hand, is the same as in the canceling case, as summarized in Table 7.2: the service completion of one of the n_1 tagged tasks increases the number of completed tagged tasks by one, accumulating up to job service completion; and other service completions decrease the queue-length observed by the tagged tasks in these queues by one. The only minor difference arises when considering the longest queues. In the canceling case, a task that completes service in one of these queues cancels all its siblings, while in the early-canceling case, a service completion triggers the task from another job to start service, and this task cancels its siblings. In either case, the result for the tagged job is a decrease by one in the lengths of all the longest queues. Finally, the transition rates in matrix $S_{\text{(full)-(not-full)}}$ are summarized in Table 7.5, which considers a service completion in one of the shortest queues when the tagged job is in a full state, triggering the tagged job to a not-full state.

To complete the PH representation of the service-time distribution, it remains to determine $\boldsymbol{s}_{\text{ser}}$, which is the stationary distribution of the initial job service phase. Different from the canceling case, to find this vector we consider the three different conditions under which a request starts service, thus

$$\boldsymbol{s}_{\text{ser}} = [\boldsymbol{s}_{\text{busy}} + \boldsymbol{s}^{\text{all}}_{\text{not-busy}} + \boldsymbol{s}^{\text{part}}_{\text{not-busy}}, \quad \mathbf{0}].$$

Notice first that the non-zero entries correspond to the full phases, as a job can only start service in phases where zero tasks have finished service, i.e., with $R(t) = k$, and service phases are ordered such that the full phases come first. The vector $\mathbf{0}$ corresponds to the not-full phases. Next, for the full phases, we consider the following three scenarios:

(1) On arrival, a job finds all the servers busy. At equilibrium, its initial service phase then has the stationary distribution of the phase $J(t)$ just after a downward jump of the age process $X(t)$, since these jumps mark the start of a new job service during the all-busy period. Following

Proposition 3, the stationary distribution of the arrival's initial service phase is therefore

$$\boldsymbol{s}_{\text{busy}} = c\gamma\boldsymbol{\alpha}_{\text{busy}}(T - S \otimes I_{m_a}),$$

where c is a normalizing constant that ensures this vector is stochastic, and γ is the probability that a job has to wait for service upon arrival, which can be obtained as in [AM01, Section 6].

(2) A job arrives during a not-all-busy period and starts service without initiating an all busy period. As already noted, more than one arrival may be necessary to initiate the all-busy period. Among these arrivals, all but one starts service with all their tasks, while the last one does initiate an all-busy period and starts service with a number of tasks equal to the number of idle servers just before it arrived. Let η_1 be the number of arrivals in a not-all-busy period, the expected value of which, $E[\eta_1]$, can be obtained from [AM01, Section 7.2], so that $E[\eta_1] - 1$ is the expected number of jobs that arrive during the not-all-busy period and find there are more than k idle servers. The probability that a job arriving during the not-all-busy period starts service with all its tasks is therefore $p_k = (E[\eta_1] - 1)/E[\eta_1]$. These arrivals start service with all their tasks, and so in the service phase with $n_0 = k$. We can write their initial service phase distribution as

$$\boldsymbol{s}^{\text{all}}_{\text{not-busy}} = (1 - \gamma)p_k[\mathbf{0}, \quad 1],$$

where we assume that the full phases are ordered increasingly with respect to n_0, so that the last phase corresponds to $n_0 = k$.

(3) A job arrives in a not-all-busy period, but finds at most k idle servers and so initiates an all-busy period, starting service with initial phase distribution

$$\boldsymbol{s}^{\text{part}}_{\text{not-busy}} = (1 - \gamma)(1 - p_k)\tilde{\boldsymbol{\pi}}(0),$$

where the components of the vector $\tilde{\boldsymbol{\pi}}(0)$ are the phases of $\boldsymbol{\pi}(0)$ that correspond to full phases, i.e., with $R(t) = k$, as $\boldsymbol{\pi}(0)$ is precisely the steady state distribution of the phase at the beginning of an all-busy period.

Table 7.6: Transition rates for matrix S_{ser}

From	To	Rate	Condition
$(n_1, \ldots, n_{m_{\text{task}}})$	$(n_1, \ldots, n_i - 1, \ldots, n_j + 1, \ldots, n_{m_{\text{task}}})$	$n_i S_{\text{task}}(i,j)$	$-$
$(n_1, \ldots, n_{m_{\text{task}}})$	$(n_1, \ldots, n_i - 1, \ldots, n_{m_{\text{task}}})$	$n_i S^*_{\text{task}}(i)$	$\sum_{i=1}^{m_{\text{task}}} n_i > r - k + 1$

We have therefore obtained the service- and waiting-time distributions, from which we can obtain the PH representation of the response-time distribution by Eq. (6.2). The resource consumption in this case is $U = k\lambda/(r\mu)$, where λ is the request arrival rate and μ^{-1} mean *task* service time, respectively. Unlike the canceling case, where we had to compute the expected number of servers used by each job, under early canceling, exactly k servers are used by every job .

7.5 Split-merge model

In this section, we consider our "baseline" model with split-(partial-)merge operation, general PH task processing times and MAP arrivals. To solve the model with PH task processing times, the key is to keep track of the phase of all tasks in service. Assuming the task processing-time distribution is $\text{PH}(\boldsymbol{s}_{\text{task}}, S_{\text{task}})$ with m_{task} phases, we define the service process of the job in service as $S^{\text{PH}}(t) = (n_1(t), \ldots, n_{m_{\text{task}}}(t))$, where $n_p(t)$ is the number of tasks in service in phase p at time t, for $1 \leq p \leq m_{\text{task}}$. Thus $S^{\text{PH}}(t)$ takes values in the set $S^{\text{PH}} = \{\boldsymbol{n} = (n_1, \ldots, n_{m_{\text{task}}}) | n_p \in \{0, \ldots, r\}, r - k + 1 \leq \sum_{p=1}^{m_{\text{task}}} n_p \leq r\}$. Note that we can represent exponential task processing times with mean processing time $1/\mu$, by setting $\boldsymbol{s}_{\text{task}} = 1$ and $S_{\text{task}} = \mu$, with $m_{\text{task}} = 1$ phase. The job service time is the interval between the time that all the job tasks enter service, and the time the k^{th} task completes service, according to the split-merge policy.

7.5.1 The job service-time distribution

Following this definition, we first seek the PH representation $\text{PH}(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ of the job service time. As all r tasks of a job start service at the same time, and each of them selects its initial

phase independently, the initial distribution of the job service phase $\boldsymbol{s}_{\text{ser}}$ follows a multinomial distribution, where the probability that a job starts service in phase $\boldsymbol{n}$ is

$$\boldsymbol{s}_{\text{ser}}(\boldsymbol{n}) = r! \prod_{i=1}^{m_{\text{task}}} \frac{\boldsymbol{s}_{\text{task}}(i)^{n_i}}{n_i!}, \quad \boldsymbol{n} \in S^{\text{PH}},$$

where $\boldsymbol{s}_{\text{task}}(i)$ is the probability that a task starts service in phase i. The transition rates of the sub-generator S_{ser} are summarized in Table 7.6. The first row considers the case where one task in service phase i jumps to phase j without completing service. The second row covers the case where a task in phase i completes service.

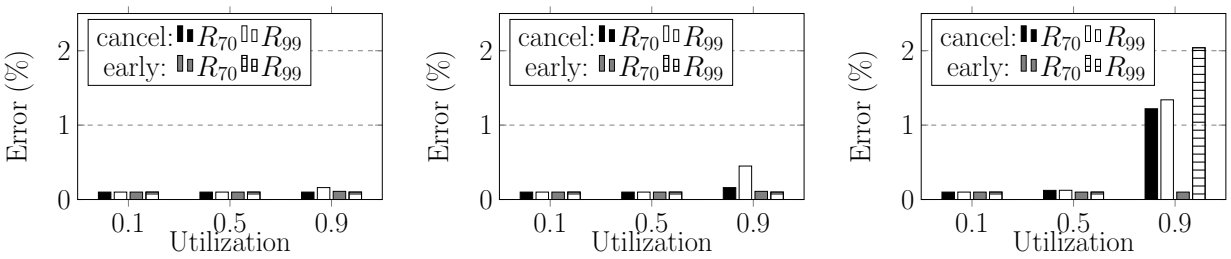
7.5.2 The job waiting-time distribution

To determine the waiting-time distribution, we again define a bivariate Markov process $\{(X(t), J^{\text{PH}}(t)|t \geq 0\}$, where the *age* $X(t)$ is the total time-in-system of the job in service, and the *phase* $J^{\text{PH}}(t) = S^{\text{PH}}(t)$ is the same as the one defined for the service process. We thus need to define matrices S and $A^{(\text{jump})}$ that describe the evolution of the service phase process. The off-diagonal entries of S hold the service transition rates without a job service completion, thus it is given by S_{ser}. Similarly, $A^{(\text{jump})} = -(S_{\text{ser}}\mathbf{1})\boldsymbol{s}_{\text{ser}}$, as the vector $-S_{\text{ser}}\mathbf{1}$ holds the rates with which a job completes service, and a new job selects its initial phase according to $\boldsymbol{s}_{\text{ser}}$. We thus follow the steps in Section 7.3 to derive the job waiting-time distribution PH $(\boldsymbol{s}_{\text{wait}}, S_{\text{wait}})$.

After obtaining the PH representations for the waiting-time and service-time distributions, we can obtain the PH representation of the response-time distribution as

$$\boldsymbol{s}_{\text{res}} = [\boldsymbol{s}_{\text{wait}} \quad (1 - \boldsymbol{\alpha}_{\text{wait}}\mathbf{1})\boldsymbol{s}_{\text{ser}}], \quad S_{\text{res}} = \begin{bmatrix} S_{\text{wait}} & (-S_{\text{wait}}\mathbf{1})\boldsymbol{s}_{\text{ser}} \\ \mathbf{0} & S_{\text{ser}} \end{bmatrix},$$

given the independence between service and waiting times. This representation captures that for some jobs the response time is made of a first period of waiting followed by a period of service, while others start service directly.



(a) (r,k)=(6,4), Exp arrivals (b) (r,k)=(9,6), Exp arrivals (c) (r,k)=(6,4), MAP arrivals

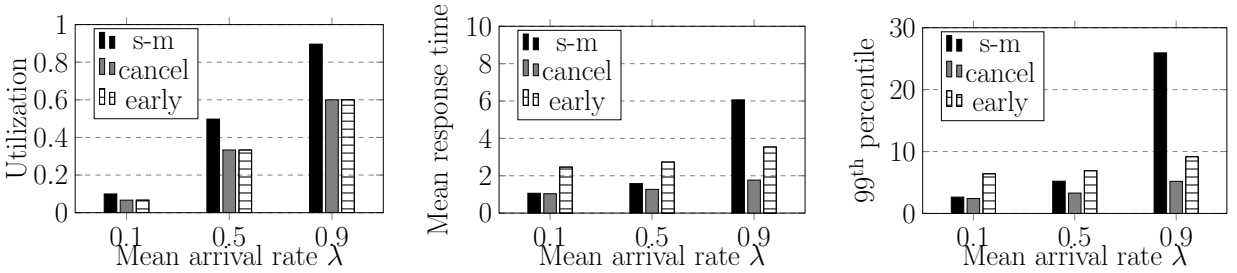
Figure 7.3: Approximation error w.r.t. simulation – $C = 10$

7.6 Numerical results

The models developed are now applied to evaluate the impact of erasure coding on delivered response time. First, the accuracy of the models is assessed by comparing against simulation.

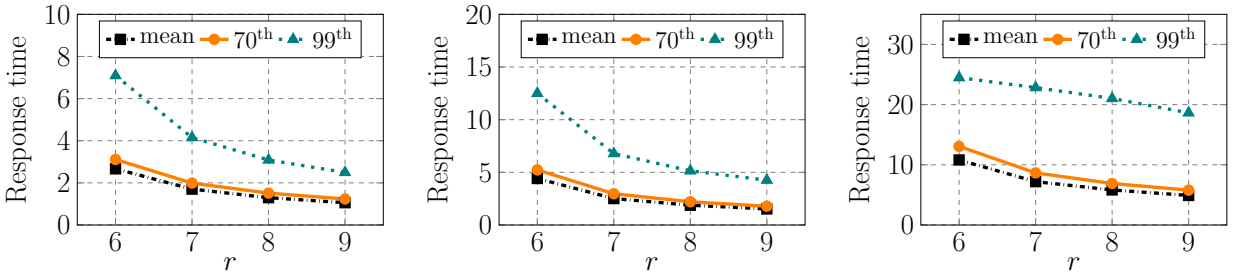
7.6.1 Validation

The accuracy of the models' approximations, i.e., the truncation of the difference between any queue and the shortest queue to a constant C, is assessed for both the canceling and early-canceling cases, considering different utilization levels, namely, low (0.1), medium (0.5), and high (0.9), for the case of $(r, k) = (6, 4)$ as an example. Assuming Poisson arrivals and $C = 10$, we report the errors obtained on the $70^{\text{th}}(R_{70})$ and $99^{\text{th}}(R_{99})$ percentile of response time in Figure 7.3(a). Note that errors lower than 0.1% are rounded to 0.1%. For each setting, the simulations were run 5,000 times with 200,000 samples each time, from which we estimated the mean response time as well as its 70^{th} and 99^{th} percentiles. We found the errors on the mean in all test cases to be below 1%; thus we focus on the 70^{th} and 99^{th} percentiles in the following. From the figure, it can be seen that, although C is relatively small, the model provides accurate results for both the 70^{th} and the 99^{th} percentile. Keeping the same C but increasing the values of r to 9 and k to 6, we show in Figure 7.3(b) the corresponding errors, which increase slightly. This is because having more queues potentially incurs a larger difference between the longest queue and the shortest one. Moreover, we observe larger errors at high utilization (0.9), although they are still lower than 3%. This is because at higher loads, the queue-length differences are greater, thus requiring a larger value of C to achieve the same



(a) Utilization (b) Mean response time (c) 99^{th} percentile response time

Figure 7.4: Comparison of three policies – $(r, k) = (9, 6)$, $C = 10$.



(a) Utilization:0.1 (b) Utilization:0.5 (c) Utilization:0.9

Figure 7.5: Comparison of the redundancy level $r - k$ (Canceling case, $k = 6$, $C = 10$).

accuracy.

Figure 7.3(c) shows the case when the Poisson arrivals in the (6,4) case are replaced by an order-2 MAP, with inter-arrival time having the same mean but a squared coefficient of variation (SCV) of 10, together with an auto-correlation function (ACF) having a decay rate of 0.5, which we obtained by the method of [HHG06]. Clearly, in this setup the model incurs larger errors compared to the case in Figure 7.3(a). This is simply because its higher variability and auto-correlation leads to burstier traffic, creating longer queues and a higher probability of having large differences between queue lengths. Accuracy can be improved, arbitrarily, by increasing the value of C, but at greater computational cost.

7.6.2 Comparing cancelation policies

In this section, we compare the response time statistics obtained under the canceling and early-canceling policies against the split-merge model. Specifically, we consider an $(r, k) = (9, 6)$ set-up with mean service time $\mu = 1$ and Poisson arrivals with rates $\lambda = 0.1$, 0.5 and 0.9. Figure 7.4 depicts the utilization, the mean and the 99^{th} percentile of response time achieved

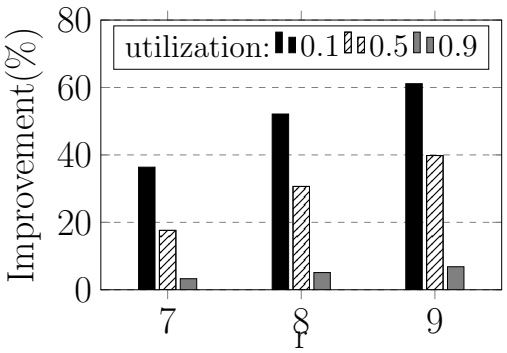


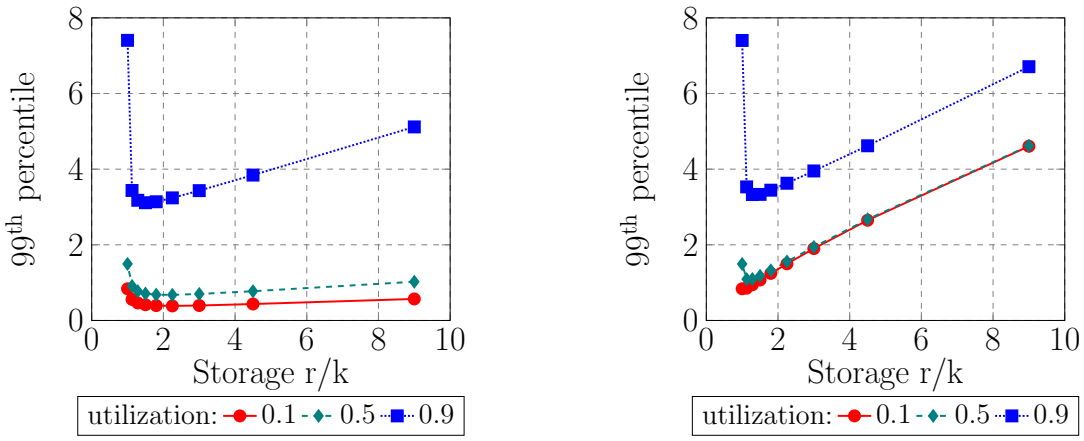
Figure 7.6: Improvement on response time's 99$^{\text{th}}$ percentile for canceling over early-canceling ($C = 10$)

under different policies. From Figure 7.4(a), we observe clearly that the utilization is highest in the split-merge case, as servers are blocked by the job in service when it has tasks outstanding at other servers. Further, we find that the with canceling case achieves the same load as the early-canceling case. This result may seems surprising at first hand, as the request in the early-canceling case uses exactly K servers while the with-canceling case uses at least K servers. The reason lies on the fact that although the canceling case uses more servers, the job service time is reduced thanks to the diversity of servers as we wait only for a subset of tasks to finish.

In terms of the response time, it can be seen that the canceling policy achieves the lowest response time, for both the mean and the 99$^{\text{th}}$ percentile, partly because it limits the extra load introduced by redundant tasks but mainly because it benefits from resource diversity. We also observe that, when the load is low, for instance when $\lambda = 0.1$, early-canceling has longer response times than even split-merge. Again, this is because the early-canceling policy cannot benefit from resource diversity as does split-merge.

7.6.3 The impact of redundancy level

Throughout this sub-section, we set $k = 6$ with mean service rate $\mu = 1$, and use different values of $r = 6$, 7, 8 and 9, with redundancy levels $r - k = 0$, 1, 2, 3, respectively. We show in Figure 7.5(a)-(c), the mean and the 70$^{\text{th}}$ and 99$^{\text{th}}$ percentiles of response time obtained for the canceling set-up, at utilizations of 0.1, 0.5 and 0.9, respectively. Clearly, all the response time metrics decrease with increasing r, at all the utilization levels. This is because a job only waits



(a) Canceling (b) Early-canceling

Figure 7.7: Trade-off between storage space r/k (corresponding to decreasing k horizontally) and 99th percentile of response time ($r = 9$, $k = \{1, \ldots, 9\}$, $C = 10$).

for the first k tasks to complete service, thus benefiting from the more diverse response times with increasing r. The largest decrease in the response time metrics occurs when r increases from 6 to 7, while additional redundant tasks have a smaller impact. Further, these gains become more limited as the utilization increases. Similar results can be observed under the early-canceling policy.

To compare the canceling and early-canceling policies, Figure 7.6 shows the percentage improvement on the 99$^{\text{th}}$ percentile of response time for the canceling case over the early-canceling case. The first observation is that the improvement decreases with the load, as we have already noted, because when the utilization is low, queuing time is small, so that any reduction in it offered by early-canceling is very limited. Instead, at low loads, coding with canceling can still benefit from the diversity in the service times, thus achieving a significant improvement over the early-canceling case. However, when the utilization increases, early-canceling has the advantage of avoiding the extra load introduced by redundant tasks under canceling, when the first k have started service. Hence the difference between the two canceling schemes becomes smaller. Further, we observe that the difference between the two schemes amplifies as r increases, as this allows canceling to exploit more diverse resources to reduce job service time. We have obtained similar results by fixing r and varying k, observing consistent superiority of the canceling set-up over early canceling, the difference decreasing with the load and the value of k. We explore this further in Section 7.6.4.

7.6.4 Storage-latency trade-off

For an object of k fragments, the storage space needed for each fragment is $1/k$ storage units (chosen so that the object has size one) and so, if the object is encoded to have r fragments, the total storage space needed is r/k units. Any gain in response time due to redundancy is therefore achieved at a cost of storage space. When the number of servers is fixed to $r = 9$, Figures 7.7(a)-(b) show the 99^{th} percentile of response time and the storage space usage for $k = \{1, \ldots, 9\}$, under canceling and early-canceling, respectively. We observe that for both policies, when the utilization is 0.5 and 0.9, the response-time percentile first decreases and then increases with the increase in k. For instance, under the canceling policy, the 99^{th} percentile of response time under 0.9 load decreases with k, reaching its lowest value at $k = 6$, i.e., $r/k = 1.5$. This is because when k is small, the sojourn time per task (including queueing time) is relatively large, leading to longer job response time; when k is large (especially when $k = r$, i.e., $r/k = 1$), waiting for many tasks to complete increases the probability of having a straggler with exceptionally long service time (e.g. a deteriorating SSD cell). However, when the load is 0.1, we observe that for the canceling case the 99^{th} percentile is very low at $k = 1$ (highest storage $r/k = 9$), close to its minimum, whereas for the early-canceling case the lowest response time is achieved using the least storage, i.e., at $k = r$ or $r/k = 1$. Thus, for $k = 9$, the canceling set-up has maximum response time, whereas the early-canceling set-up achieves the lowest. To provide some insight into these observations, we use the following lemmas, which apply at low utilization where job queuing times are least.

Lemma 7.6.1. *The minimum mean service time under canceling is achieved when $k = 1$.*

Proof. In low loads, the response times are mostly composed by the service times as queuing times are negligible. If we focus on the service time only, under canceling this time comprises k phases, where in phase $1 \leq i \leq k$ there are $r - i + 1$ parallel tasks in service. In each stage the mean service time is $1/(k\mu)$, assuming a uniform division of a job into tasks that are each k times shorter, on average, than in the single processor case. The mean service time for a job

can therefore be written as

$$E[S_C(k,r)] = \frac{1}{k\mu}\sum_{i=1}^{k}\frac{1}{r-i+1},$$

where $S_C(k,r)$ is the service time random variable for a job under canceling with (r,k) coding. This can be rewritten as, after some rearrangement,

$$E[S_C(k+1,r)] + \frac{1}{(k+1)\mu}\left(\frac{1}{k}\sum_{i=1}^{k}\frac{1}{r-i+1} - \frac{1}{r-k}\right).$$

Each term in the sum is strictly less than $1/(r-k)$ so that $E[S_C(k,r)] < E[S_C(k+1,r)]$ for all k. Hence the minimum mean service time is obtained when $k=1$. □

Lemma 7.6.2. *The minimum mean service time for the early-canceling policy is achieved when* $k = r$.

Proof. As in the canceling case, with early-canceling, the service time S_E comprises k phases, where in phase i there are $k-i+1$ tasks in service. Again, a task has mean service time $1/(k\mu)$ so that the mean job service time is

$$E[S_E(k,r)] = \frac{1}{k\mu}\sum_{i=1}^{k}\frac{1}{k-i+1} = \frac{1}{k\mu}\sum_{i=1}^{k}\frac{1}{i},$$

This can be written as

$$E[S_E(k+1,r)] + \frac{1}{(k+1)\mu}\left(\frac{1}{k}\sum_{i=1}^{k}\frac{1}{i} - \frac{1}{k+1}\right).$$

Each term in the sum is less than 1 so that $E[S_C(k,r)] > E[S_C(k+1,r)]$ for all k. The minimum mean service time is therefore obtained when $k=r$. □

These results indicate that at low loads, where the service time is the dominant component of response time, it is optimal to select $k=1$ for canceling and $k=r$ for early canceling. This is consistent with what we observed above when considering percentiles as the objective rather

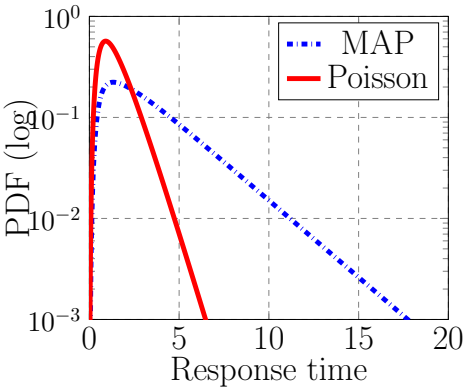


Figure 7.8: Response-time PDF under Possion and MAP arrivals; (r,k)=(6,4), utilization 0.5, $C = 10$.

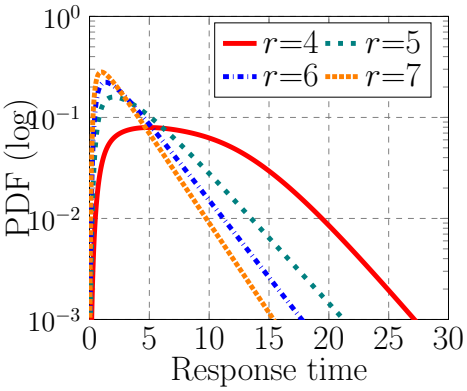


Figure 7.9: Response-time PDF ($k = 4$, MAP arrivals, utilization 0.5, $r = 4, 5, 6, 7$, $C = 10$).

Table 7.7: Response-time percentiles and the improvement (%) over $r = 4$

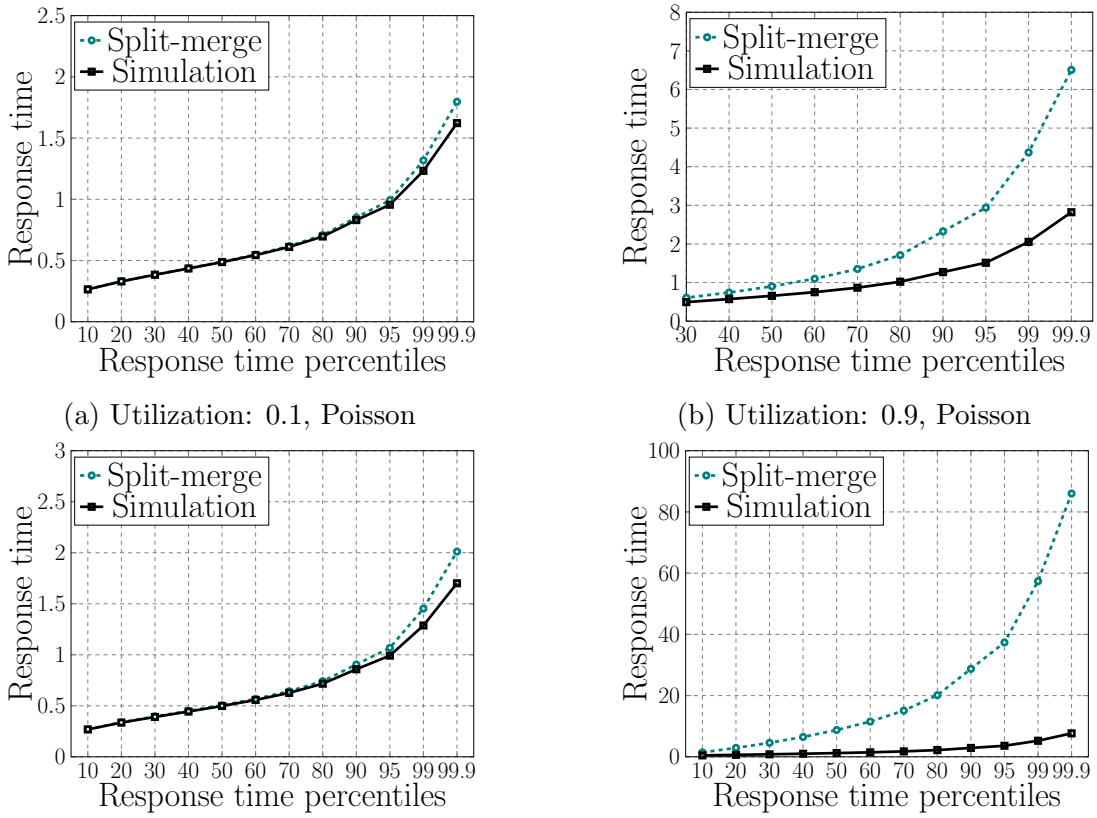
Percentile	**r=4**	**r=5**	**r=6**	**r=7**
70^{th}	10.78	5.96 (44.71%)	4.43 (58.91%)	3.61 (66.51%)
95^{th}	18.10	12.17 (32.76%)	9.57 (47.13%)	7.99 (55.86%)
99^{th}	23.48	17.57 (25.17%)	14.16 (39.69%)	11.93 (49.19%)
99.9^{th}	30.81	25.27 (17.98%)	20.72 (32.75%)	17.56 (43.01%)

than the mean. When the load increases, queuing times become more significant but coding reduces queuing time with increasing k, i.e., as the size of tasks decreases.

7.6.5 MAP arrivals

We now consider the MAP of Section 7.6.1, with SCV 10 and ACF decay-rate 0.5. To be concrete, we consider the canceling set-up with $(r, k) = (6, 4)$ at 0.5 utilization. Figure 7.8 shows the response-time PDFs obtained under both Poisson and MAP arrivals, and we note a much longer tail in the latter case. This suggests that coding might be more beneficial with highly-variable and auto-correlated arrivals in tackling a long tail.

Figure 7.9 shows the response-time PDF obtained for $k = 4$ and $r = 4, 5, 6, 7$. We see that even a redundancy of just 1 (i.e. $r = k + 1$) is highly effective in shortening the response-time tail. Table 7.7 shows the 70^{th}, 95^{th}, 99^{th} and 99.9^{th} percentiles for different values of r, and the corresponding improvement that r =5, 6, 7 have over the $r = 4$ case. We note large reductions in the response-time tails caused by the coding scheme.



(a) Utilization: 0.1, Poisson

(b) Utilization: 0.9, Poisson

(c) Utilization: 0.1, MAP

(d) Utilization: 0.9, MAP

Figure 7.10: The split-merge model provides an upper bound on the canceling model – (r,k)=(9,6), $C = 10$.

7.6.6 Canceling with general processing times

The PH distribution is a good candidate for modeling processing times with different levels of variability. However, models quickly become computationally expensive since the size of the phase space increases exponentially with the number of task service phases. We therefore turn to an approximate approach to determine bounds on the response time for the canceling set-up, which we choose in view of its superior performance over the early canceling policy.

An upper bound is provided by the split-merge model, where servers are blocked by a single job until any k out of its r tasks finish service, thus introducing a synchronization point that makes the analysis more tractable. Note that [JSW15] also uses this model to derive an upper bound on the *mean* response time, whereas our split-merge model is able to provide the full response-time *distribution*.

We show in Figure 7.10(a)-(b) and Figure 7.10(c)-(d) the response-time percentiles obtained

by the split-merge model with Poisson arrivals and MAP arrivals (SCV = 10, decay-rate = 0.5), respectively, alongside simulation results, at utilization levels 0.1 and 0.9. For the task processing-time, we used a mixture of two random variables: one negative exponential and one Erlang-4, with respective means 0.5 and 50.5 units and weights 99% and 1%, giving an overall mean service time of 1. As expected, we observe tight bounds when the utilization is low, as the response time comprises mostly service times and the impact of server-blocking is small. However, the bounds become looser with increasing utilization, as the queuing time becomes significant at high loads due to the server blocking, thus providing a very conservative bound at utilization 0.9. Further, it is worth noting that the bound is looser on the tail than on the lower percentiles. Also, with MAP arrivals, we observe a large difference between the bound and the simulation results at almost all percentiles, as bursty arrivals lead to longer queuing times, which are amplified due to the blocking effect in the split-merge model.

7.7 Discussion

We have evaluated two policies for managing (r, k) partial-join queues and compared them with a "baseline", split-merge policy in terms of job response-time distribution. In general, long response times can occur when service times jump up, when arrivals become more intense, or when both occur roughly at the same time. These effects are compounded when a job has to wait for all its parallel tasks. This is common in erasure encoded storage systems but consistent performance is still required as the system ages and cells deteriorate electronically in SSDs. Some of the conclusions we draw from our experimental results are the following:

- In (r, k) partial-join systems with MAP arrivals and exponential service times, both canceling policies significantly reduce the high quantiles of job response time, especially with highly variable and correlated MAP arrival streams.

- Canceling achieves the lowest response times across the range of utilizations since it benefits from the property of resource diversity, whilst also limiting the extra load introduced.

- Early canceling, on the other hand, is easier to implement as in-queue canceling requires less modification to the system than canceling in-service. The choice, which is also affected by the utilization, can be made on a sound, quantitative basis, in terms of performance improvement and cost.

- Job response-time metrics decrease with increasing r, at all utilization levels, for both the canceling and early-canceling schemes. However, any gain in response time is achieved at a cost of storage space: if an object is encoded to have r fragments, the total storage space needed is r/k units - i.e. more than 1.

- When considering general PH processing times, the split-merge model simplifies the analysis and provides an upper bound on the response times. The bounds are tight when the utilization is low, however, become looser with increasing utilization. Further, it is worth noting that the bound is looser under bursty arrivals, and looser on the tail than on the lower percentiles.

Chapter 8

Conclusion

8.1 Summary of Thesis Achievements

Modern computer systems provide users with cost-effective and high-performance services by exploiting large collections of inexpensive resources. However, the increasing scale and complexity of these systems makes it challenging to achieve high reliability and consistently low response times. In this thesis, we consider request replication as a strategy to improve both reliability and request latency, by issuing r replicas of each request and using the result of the first k ($1 \leq k \leq r$) replicas to complete. Request replication is effective to handle unpredictable failures, with the potential to reduce response times, even at high utilization levels. However, it introduces additional load that may negatively impact latency and lead to excessive delays. To capture the *trade-off* between these two conflicting effects of replication, and characterize the scenarios under which replication improves latency, we introduced stochastic models to evaluate how replication affects the service level objectives in various systems.

We first focused on the case with $k = 1$, where the system replies with the result from whichever replica completes successfully first. In Chapters 3 and 4 we proposed stochastic models to determine the response-time distribution for systems where requests are subject to failures, where both service and failure times are exponentially distributed. We made use of the proposed models to compare the distributed set-up against the centralized case, where we found that

while the centralized case achieves slightly shorter latency, the distributed set-up provides more flexibility, making it easier to increase or decrease the number of servers as the utilization rises or falls. In Section 4.4 we also extended our analysis to consider parallel jobs that are composed of multiple tasks, where the job response time is determined by the last task to finish. We then considered in Chapter 5 reliable systems with general PH request processing times, which allow us to capture the variability observed in the server side. We summarize our key findings regarding replication in the following.

1. Replication is effective in reducing latency, particularly the tail of the latency distribution. However, this depends on a number of factors, including the canceling policy, the arrival and service processes, the utilization, and the number of replicas. In particular, service time variability plays a key role when evaluating the effect of replication on latency. Highly variable service times provide larger gains, and offer more chances to benefit from replication.

2. The effect of replication is nonuniform over the response-time distribution. As a result, a decision to replicate based solely on potential gains on one percentile may actually have a negative effect on the others. This highlights the importance of explicitly considering the response-time distribution, and the targeted percentiles, when evaluating a replication strategy.

3. The threshold utilization is nonuniform along the response-time distribution. It increases with the number of servers, but decreases with the variability and auto-correlation of the arrival process.

In the second part of this thesis, we considered fork-join queues with $k = r$, as described in Chapter 6. We proposed an approximate approach that is able to evaluate the response-time percentiles, as well as to accommodate phase-type processing times and Markovian arrival processes. Extensive validation reveals that the proposed approach is very accurate, with the advantage of being efficient and scalable.

Finally, in Chapter 7 we dealt with general order statistics of request response times for systems that implement *erasure encoding*. We evaluated both the canceling and early-canceling policies

for managing (r, k) partial-join queues, and compared them with a baseline model – split-merge policy in terms of job response-time distribution. Experimental results show that long response times can occur when service times jump up, when arrivals become more intense, or when both occur roughly at the same time.

In summary, different from existing works, which focus on the mean response time as the sole performance metric, the models introduced in this thesis are able to determine the response-time distribution, which enable to evaluate the impact of replication on different response-time percentiles, which, as we demonstrated, is far from homogeneous. We made use of the models to find the threshold utilization, i.e., the maximum utilization under which replication is beneficial, which provides the guidance on when to adopt replication in real applications.

8.2 Future work

Request replication has been widely considered recently as a powerful mechanism to improve reliability, and to limit the response time. In this thesis, we have provided the theoretical guidance on when to adopt replication, and how many replicas to adopt. There are many potential avenues for future work, among which those of particular interest are described below.

Real application experiments We validated the accuracy of the models with replication without canceling in Section 5.2, via extensive evaluations on a MATLAB benchmark and MediaWiki application. However, deploying replication with canceling in real applications, remains to be challenging as it involves modifying the system scheduling policy, enabling the communication between servers, and implementing the canceling policy. We are curious to see how this simple but impactful idea works in various applications in the future.

Heterogeneous servers In this thesis we considered homogeneous servers, i.e., we assumed that the systems under consideration are consisted of a number of homogeneous and independent servers. Future work may consider the extension of the models to consider heterogeneous servers, which will lead to much more complicated models with larger state space.

Efficient computation A limitation of the matrix-analytic method lies in the fact that it operates on matrices that can grow very fast in size with the number of servers, especially if the processing-time distribution has many phases, the scale of the system is too large, and if the system gets too complicated. For example, as shown in Chapter 7, when considering the general PH distributed processing times, the models proposed quickly become computationally intractable. While we have devised efficient methods to obtain the response-time metrics, future work will look into alternative approaches to tackle large-scale systems.

Bibliography

[AGSS12] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Why let resources idle? aggressive cloning of jobs with dolly. In *Memory*, 2012.

[AKG+10] Ganesh Ananthanarayanan, Srikanth Kandula, Albert G Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in Map-Reduce clusters using mantri. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, volume 10, page 24, 2010.

[AM01] Søren Asmussen and Jakob R Møller. Calculation of the steady state waiting time distribution in GI/PH/c and MAP/PH/c queues. *Queueing Systems*, 37:9–29, 2001.

[ANO96] Søren Asmussen, Olle Nerman, and Marita Olsson. Fitting phase-type distributions via the EM algorithm. *Scandinavian Journal of Statistics*, pages 419–441, 1996.

[BDVD98] Simonetta Balsamo, Lorenzo Donatiello, and Nico M Van Dijk. Bound performance models of heterogeneous parallel processing systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):1041–1056, 1998.

[BHT05] Andrea Bobbio, András Horváth, and Miklós Telek. Matching three moments with minimal acyclic phase type distributions. *Stochastic models*, 21(2-3):303–326, 2005.

[BL11] Christian Bienia and Kai Li. *Benchmarking modern multiprocessors*. Princeton University New York, 2011.

[Bla15] Simple benchmarking of parfor using Blackjack. `http://uk.mathworks.com/help/distcomp/examples/simple-benchmarking-of-parfor-using-blackjack.html`, 2015.

[BLM05] Dario A Bini, Guy Latouche, and Beatrice Meini. *Numerical methods for structured Markov chains*. Oxford University Press on Demand, 2005.

[BMSVH06] DA Bini, B Meini, S Steffé, and B Van Houdt. Structured Markov chains solver: software tools. In *Workshop on Tools for solving structured Markov chains*, page 14. ACM, 2006.

[Bru09] Jake Brutlag. Speed matters for Google web search. *Google. June*, 2009.

[CFG+05] Zizhong Chen, Graham E Fagg, Edgar Gabriel, Julien Langou, Thara Angskun, George Bosilca, and Jack Dongarra. Fault tolerant high performance computing by a coding approach. In *ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 213–223. ACM, 2005.

[CSH+14] Shengbo Chen, Yin Sun, Longbo Huang, Prasun Sinha, Guanfeng Liang, Xin Liu, Ness B Shroff, et al. When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1042–1050. IEEE, 2014.

[Cum82] Aldo Cumani. On the canonical representation of homogeneous Markov processes modelling failure-time distributions. *Microelectronics Reliability*, 22(3):583–602, 1982.

[DA00] Jeffrey E Diamond and Attahiru Sule Alfa. On approximating higher order MAPs with MAPs of order two. *Queueing Systems*, 34(1-4):269–288, 2000.

[DB13] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38, 1977.

[DN81] Herbert Aron David and Haikady Navada Nagaraja. *Order statistics*. Wiley Online Library, 1981.

[FH84] Leopold Flatto and S Hahn. Two parallel queues created by arrivals with two demands I. *SIAM Journal on Applied Mathematics*, 44(5):1041–1053, 1984.

[FJV+12] Benjamin Farley, Ari Juels, Venkatanathan Varadarajan, Thomas Ristenpart, Kevin D Bowers, and Michael M Swift. More for your money: exploiting performance heterogeneity in public clouds. In *ACM Symposium on Cloud Computing*, page 20. ACM, 2012.

[FSLI+11] Kurt Ferreira, Jon Stearley, James H Laros III, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Rolf Riesen, Patrick G Bridges, and Dorian Arnold. Evaluating the viability of process replication reliability for exascale systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2011.

[GHBDL09] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 157–168. ACM, 2009.

[GHBDZ10] Varun Gupta, Mor Harchol-Balter, JG Dai, and Bert Zwart. On the inapproximability of M/G/K: why two moments of job size distribution are not enough. *Queueing Systems*, 64(1):5–48, 2010.

[GHBSW16] K. Gardner, M. Harchol-Balter, and A. Scheller-Wolf. A better model for job redundancy: Decoupling server slowdown and job size. In *MASCOTS*, 2016.

[GJL84] DP Gaver, PA Jacobs, and Guy Latouche. Finite birth-and-death models in randomly changing environments. *Advances in applied probability*, pages 715–731, 1984.

[GJW11] Brian Guenter, Navendu Jain, and Charles Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1332–1340. IEEE, 2011.

[GNVL79] Gene Golub, Stephen Nash, and Charles Van Loan. A Hessenberg-Schur method for the problem AX+XB=C. *IEEE Transactions on Automatic Control*, 24(6):909–913, 1979.

[Gum54] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*. Number 33. US Govt. Print. Office, 1954.

[GZD+15] Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, and Esa Hyytia. Reducing latency via redundant requests: Exact analysis. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):347–360, 2015.

[Hag89] William W Hager. Updating the inverse of a matrix. *SIAM review*, 31(2):221–239, 1989.

[He12] Qiming He. Analysis of a continuous time SM[K]/PH [K]/1/FCFS queue: Age process, sojourn times, and queue lengths. *Journal of Systems Science and Complexity*, 25(1):133–155, 2012.

[Hei04] Armin Heindl. Inverse characterization of hyperexponential MAP(2)s. In *Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA)*, volume 31. Citeseer, 2004.

[HHG06] Armin Heindl, Gábor Horváth, and Karsten Gross. Explicit inverse characterizations of acyclic maps of second order. In *European Performance Engineering Workshop (EPEW)*, pages 108–122. Springer, 2006.

[Hor90] Roger A Horn. The Hadamard product. In *Proceedings of Symposia in Applied Mathematics*, volume 40, pages 87–169, 1990.

[HP92] Peter G Harrison and Naresh M Patel. *Performance Modelling of Communication Networks and Computer Architectures (International Computer S.* Addison-Wesley Longman Publishing Co., Inc., 1992.

[HP93] Peter G Harrison and Edwige Pitel. Sojourn times in single-server queues with negative customers. *Journal of Applied Probability*, pages 943–963, 1993.

[HPZR12] Longbo Huang, Sameer Pawar, Hao Zhang, and Kannan Ramchandran. Codes can reduce queueing delay in data centers. In *IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 2766–2770. IEEE, 2012.

[HQ13] Peter G Harrison and Zhan Qiu. Performance enhancement by means of task replication. In *European Workshop on Performance Engineering (EPEW)*, pages 191–205. Springer, 2013.

[htt15] httperf. `https://github.com/httperf/httperf`, 2015.

[HZ07] Peter Harrison and Soraya Zertal. Queueing models of raid systems with maxima of waiting times. *Performance Evaluation*, 64(7):664–689, 2007.

[JLS14] Gauri Joshi, Yanpei Liu, and Emina Soljanin. On the delay-storage trade-off in content download from coded distributed storage systems. *IEEE Journal on Selected Areas in Communications*, 32(5):989–997, 2014.

[JSW15] Gauri Joshi, Emina Soljanin, and Gregory Wornell. Efficient replication of queued tasks to reduce latency in cloud systems. *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2015.

[KA89] Cheeha Kim and Ashok K Agrawala. Analysis of the fork-join queue. *IEEE Transactions on Computers*, 38(2):250–255, 1989.

[KB03] Arie Keren and Amnon Barak. Opportunity cost algorithms for reduction of i/o and interprocess communication overhead in a computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, 14(1):39–50, 2003.

[Kie53] Jack Kiefer. Sequential minimax search for a maximum. *Proceedings of the American mathematical society*, 4(3):502–506, 1953.

[KMR08] Gopi Kandaswamy, Anirban Mandal, and Daniel A Reed. Fault tolerance and recovery of scientific workflows on computational grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 777–782. IEEE, 2008.

[KS+99] Sungyeol Kang, Richard F Serfozo, et al. Extreme values of phase-type and mixed random variables with parallel-processing examples. *Journal of applied probability*, 36(1):194–210, 1999.

[KS04] Sung-Seok Ko and Richard F Serfozo. Response times in M/M/s fork-join networks. *Advances in Applied Probability*, pages 854–871, 2004.

[KS08] Sung-Seok Ko and Richard F Serfozo. Sojourn times in G/M/1 fork-join networks. *Naval Research Logistics (NRL)*, 55(5):432–443, 2008.

[KSH03] Rachid El Abdouni Khayari, Ramin Sadre, and Boudewijn R Haverkort. Fitting world-wide web request traces with the EM-algorithm. *Performance Evaluation*, 52(2):175–191, 2003.

[Lin06] Greg Linden. Marissa mayer at web 2.0. `http://glinden.blogspot.com/2006/11/marissa-mayer-atweb-20.html`, 2006.

[LK07] Abigail S Lebrecht and William J Knottenbelt. Response time approximations in fork-join queues. In *UK Performance Engineering Workshop (UKPEW)*, 2007.

[LK14] Guanfeng Liang and Ulaş C Kozat. Fast cloud: Pushing the envelope on delay performance of cloud storage with coding. *IEEE/ACM Transactions on Networking*, 22(6):2012–2025, 2014.

[LMT94] John Chi-Shing Lui, Richard R Muntz, and Don Towsley. *Computing performance bounds for fork-join queueing models*. University of California (Los Angeles). Computer Science Department, 1994.

[LPR15] K. Lee, R. Pedarsani, and K. Ramchandran. On scheduling redundant requests with cancellation overheads. In *Allerton*, 2015.

[LR99] Guy Latouche and Vaidyanathan Ramaswami. *Introduction to matrix analytic methods in stochastic modeling*, volume 5. Siam, 1999.

[med15] Mediawiki. `https://www.mediawiki.org/wiki/MediaWiki`, 2015.

[Mit02] Isi Mitrani. Spectral expansion solutions for markov-modulated queues. In *IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation (PERFORMANCE)*, pages 17–35. Springer, 2002.

[Mit13] Isi Mitrani. Managing performance and power consumption in a server farm. *Annals of Operations Research*, 202(1):121–134, 2013.

[Neu81] Marcel F Neuts. *Matrix-geometric solutions in stochastic models: an algorithmic approach*. Courier Corporation, 1981.

[NT88] Randolph Nelson and Asser N Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739–743, 1988.

[OAT+07] Ron A Oldfield, Sarala Arunagiri, Patricia J Teller, Seetharami Seelam, Maria Ruiz Varela, Rolf Riesen, and Philip C Roth. Modeling the impact of checkpoints on next-generation systems. In *IEEE Conference on Mass Storage Systems and Technologies (MSST)*, pages 30–46. IEEE, 2007.

[ÖV96] M Tamer Özsu and Patrick Valduriez. Distributed and parallel database systems. *ACM Computing Surveys (CSUR)*, 28(1):125–128, 1996.

[Par] Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[PGK88] David A Patterson, Garth Gibson, and Randy H Katz. *A case for redundant arrays of inexpensive disks (RAID)*, volume 17. ACM, 1988.

[PR06] Juan F Pérez and Germán Riano. jPhase: an object-oriented tool for modeling phase-type distributions. In *ACM SMCtools*, 2006.

[PVVVH08] Juan F Pérez, Jeroen Van Velthoven, and Benny Van Houdt. Q-MAM: a tool for solving infinite queues using matrix-analytic methods. In *International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*, page 16. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[QP14] Zhan Qiu and Juan F Pérez. Assessing the impact of concurrent replication with canceling in parallel jobs. In *IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 31–40. IEEE, 2014.

[QP15a] Zhan Qiu and Juan F Pérez. Enhancing reliability and response times via replication in computing clusters. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1355–1363. IEEE, 2015.

[QP15b] Zhan Qiu and Juan F Pérez. Enhancing reliability and response times via replication in computing clusters. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1355–1363. IEEE, 2015.

[QP15c] Zhan Qiu and Juan F Pérez. Evaluating the effectiveness of replication for tail-tolerance. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 443–452. IEEE, 2015.

[QP16] Zhan Qiu and Juan Perez. Evaluating replication for parallel jobs: an efficient approach. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 8(27):2288–2302, 2016.

[QPH15] Zhan Qiu, Juan F Pérez, and Peter G Harrison. Beyond the mean in fork-join queues: Efficient approximation for response-time tails. *Performance Evaluation*, 91:99–116, 2015.

[QPH16a] Zhan Qiu, Juan F Pérez, and Peter G Harrison. Tackling latency via replication in distributed systems. In *ACM/SPEC International Conference on Performance Engineering (ICPE)*, pages 197–208. ACM, 2016.

[QPH16b] Zhan Qiu, Juan F Pérez, and Peter G Harrison. Variability-aware request replication for latency curtailment. 2016.

[Ram90] V Ramaswami. A duality theorem for the matrix paradigms in queueing theory. *Stochastic Models*, 6(1):151–161, 1990.

[RAWY15] Xiaoqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. *ACM SIGCOMM Computer Communication Review*, 45(4):379–392, 2015.

[RPQ+17] Birke Robert, Juan F Pérez, Zhan Qiu, Mathias Bjorkqvist, and Lydia Y Chen. Power of redundancy: Designing partial replication for multi-tier applications. 2017.

[SDQR10] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, 3(1-2):460–471, 2010.

[Sen89] Bhaskar Sengupta. Markov processes whose steady state distribution is matrix-exponential with an application to the GI/PH/1 queue. *Advances in Applied Probability*, pages 159–180, 1989.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[SLR12] Nihar B Shah, Kangwook Lee, and Kannan Ramchandran. The MDS queue: Analysing latency performance of codes and redundant requests. Technical report, Technical Report, 2012.

[SLR16] Nihar B Shah, Kangwook Lee, and Kannan Ramchandran. When do redundant requests reduce latency? *IEEE Transactions on Communications*, 64(2):715–722, 2016.

[Sny10] Bill Snyder. Server virtualization has stalled, despite the hype. `http://www.infoworld.com/print/146901`, 2010.

[TBT06] Axel Thummler, Peter Buchholz, and Miklos Telek. A novel approach for phase-type fitting with the EM algorithm. *IEEE Transactions on Dependable and Secure Computing*, 3(3):245–258, 2006.

[Tho15] Alexander Thomasian. Analysis of fork/join and related queueing systems. *ACM Computing Surveys (CSUR)*, 47(2):17, 2015.

[VGM+13] Ashish Vulimiri, Philip Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. Low latency via redundancy. In *ACM conference on Emerging networking experiments and technologies (CoNEXT)*, pages 283–294. ACM, 2013.

[VM94] Subir Varma and Armand M Makowski. Interpolation approximations for symmetric fork-join queues. *Performance Evaluation*, 20(1-3):245–265, 1994.

[VMGS12] Ashish Vulimiri, Oliver Michel, P Godfrey, and Scott Shenker. More is less: reducing latency via redundancy. In *ACM Workshop on Hot Topics in Networks (HotNets)*, pages 13–18. ACM, 2012.

[Whi82] Ward Whitt. Approximating a point process by a renewal process, I: Two basic methods. *Operations Research*, 30(1):125–147, 1982.

[WSJ07] Ming Wu, Xian-He Sun, and Hui Jin. Performance under failures of high-end computing. In *ACM/IEEE conference on Supercomputing*, page 48. ACM, 2007.

[XBNJ13] Yunjing Xu, Michael Bailey, Brian Noble, and Farnam Jahanian. Small is better: Avoiding latency traps in virtualized data centers. In *Annual Symposium on Cloud Computing (SOCC)*, page 7. ACM, 2013.

[XLAC14] Yu Xiang, Tian Lan, Vaneet Aggarwal, and Yih Farn R Chen. Joint latency and cost optimization for erasurecoded data center storage. *ACM SIGMETRICS Performance Evaluation Review*, 42(2):3–14, 2014.

[ZL09] Ziming Zheng and Zhiling Lan. Reliability-aware scalability models for high performance computing. In *IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*, pages 1–9. IEEE, 2009.

Appendix A

A.1 Proofs

A.1.1 Proof of Proposition 4

Proof. Since jobs only fail during service, the failure probability is given by

$$p_S = \int_0^\infty \boldsymbol{s}_{\text{ser}} \exp(S_{\text{ser}} x) \boldsymbol{S}_S^* dx = -\boldsymbol{s}_{\text{ser}} S_{\text{ser}}^{-1} \boldsymbol{S}_S^*,$$

thus the probability that a job service time lasts for at most x time units and succeeds is

$$F_s(x) = \frac{1}{p_S} \int_0^x \boldsymbol{s}_{\text{ser}} \exp(S_{\text{ser}} y) \boldsymbol{S}_S^* dy = 1 + \frac{1}{p_S} \boldsymbol{s}_{\text{ser}} S_{\text{ser}}^{-1} \exp(S_{\text{ser}} x) \boldsymbol{S}_S^*.$$

This matrix-exponential representation of the service times for successful jobs can be turned into a PH representation by defining a diagonal matrix Π such that $\Pi \mathbf{1} = \eta'$, where $\eta = -\boldsymbol{s}_{\text{ser}} S_{\text{ser}}^{-1}$ is the stationary distribution of the service phase. Defining $B_{\text{ser}} = \Pi^{-1} S'_{\text{ser}} \Pi$, we have

$$\begin{aligned} 1 - F_s(x) &= -\frac{1}{p_S} \boldsymbol{s}_{\text{ser}} S_{\text{ser}}^{-1} \Pi^{-1} \Pi \exp(S_{\text{ser}} x) \Pi^{-1} \Pi \boldsymbol{S}_S^* \\ &= \frac{1}{p_S} \eta \Pi^{-1} \exp(B'_{\text{ser}} x) \Pi \boldsymbol{S}_S^* \\ &= \frac{1}{p_S} \boldsymbol{S}_S^{*\prime} \Pi \exp(B_{\text{ser}} x) \mathbf{1}. \end{aligned}$$

This defines a proper PH distribution as the vector $\boldsymbol{S}_S^{*\prime}\Pi/p_S$ is stochastic. This results from $\boldsymbol{S}_S^*$ and Π being non-negative, and $\boldsymbol{S}_S^{*\prime}\Pi\mathbf{1} = -\boldsymbol{S}_S^{*\prime}(\boldsymbol{s}_{\text{ser}}S_{\text{ser}}^{-1})' = p_S$. □

A.1.2 Proof of Theorem 3.1

Proof. To obtain the response-time distribution we first recall that the PH representation of the waiting-time distribution is obtained from a time-reversal argument [Sen89]. Thus, we rely on a time-reversal argument to build the PH representation of the response time. As in Theorem 4, we can obtain the PH representation of the service time distribution of successful jobs. Here we split this representation for jobs that wait and jobs that do not. Jobs that wait start service with phase according to $\gamma\boldsymbol{\pi}_{\text{busy}}$, thus we define the diagonal matrix Δ_{busy} such that $\Delta_{\text{busy}}\mathbf{1} = -\gamma(\boldsymbol{\pi}_{\text{busy}}S_{\text{ser}}^{-1})'$ since $\boldsymbol{\pi}_{\text{busy}}S_{\text{ser}}^{-1}$ is the stationary distribution of the service phase for jobs that wait. We then obtain the PH representation for jobs that wait $(\boldsymbol{\beta}_{\text{ser}}^{\text{busy}}, B_{\text{ser}}^{\text{busy}})$ as

$$\boldsymbol{\beta}_{\text{ser}}^{\text{busy}} = -\boldsymbol{t}_S'\Delta_{\text{busy}}/p_S, \quad B_{\text{ser}}^{\text{busy}} = \Delta_{\text{busy}}^{-1}S_{\text{ser}}'\Delta_{\text{busy}}.$$

A similar result is obtained for jobs that do not wait, considering that in this case jobs start service according to $(1-\gamma)\boldsymbol{\pi}_0$. Eq. (3.15) is built by putting together the paths of jobs that do not wait, which start service with $\boldsymbol{\beta}_{\text{ser}}^{\text{idle}}$, with those of jobs that wait, which start service with $\boldsymbol{\beta}_{\text{ser}}^{\text{busy}}$.

For jobs that wait, their response time is composed of a first stage of waiting followed by a second stage of service. Further, the phase in which the waiting stage ends determines the stage in which the service stage begins. Since the waiting-time process is reversed in time, we build the response time by first starting with a time-reversed version of the service time, which ends in a phase that determines the phase in which the time-reversed version of the waiting-time process starts. That $(\boldsymbol{\beta}_{\text{ser}}^{\text{busy}}, B_{\text{ser}}^{\text{busy}})$ is in fact the time-reversed version of the service time of successful comes from the fact that the starting vector $\boldsymbol{\beta}_{\text{ser}}^{\text{busy}} = -\boldsymbol{t}_S'\Delta_{\text{busy}}/p_S$ is a transformation of the original exit vector t_S, and the transformation matrix Δ_{busy} has diagonal elements proportional to $(\boldsymbol{\pi}_{\text{busy}}S_{\text{ser}}^{-1})$, which s the stationary distribution of the phase for jobs that wait.

To find how the phase transitions from the end of the service stage to the beginning of the waiting stage, in the time-reversed process, we observe that the joint probability that after a downward jump the level is at least x and the phase is j is given by

$$W(x) = \int_x^\infty \pi(y) \int_0^\infty \exp(Tu) A^{(\mathrm{MAP})} du dy,$$

as this considers the probability that any level greater than x is visited by means of a downward jump of any size u. From (2.9) we know that the inner integral is equal to $T - S$, obtaining

$$W(x) = \int_x^\infty \pi(y)(T-S)dy = -\boldsymbol{\pi}(0)T^{-1}\exp(Tx)(T-S) = \boldsymbol{\alpha}_{\mathrm{busy}}\exp(Tx)(T-S).$$

Now we can perform the similarity transform in [Sen89] to obtain the PH representation of the waiting-time process. We thus define a diagonal matrix Λ such that $\Lambda\mathbf{1} = \boldsymbol{\alpha}'_{\mathrm{busy}}$ to obtain

$$W(x) = \boldsymbol{\alpha}_{\mathrm{busy}}\Lambda^{-1}\Lambda\exp(Tx)\Lambda^{-1}\Lambda(T-S) = \mathbf{1}'\exp(S'_{\mathrm{wait}}x)\Lambda(T-S) = (T-S)'\Lambda\exp(S_{\mathrm{wait}}x)\mathbf{1}.$$

Notice that we use $S_{\mathrm{wait}} = \Lambda T'\Lambda^{-1}$, which is equivalent to the definition in Eq. (2.18). We therefore have the standard representation for the sub-generator of the waiting-time distribution, but the term $(T-S)'\Lambda$ captures the transition from the final phase in the time-reversed service process to the initial phase in the time-reversed waiting-time process. We now need to normalize this by evaluating $W(x)$ at $x = 0$, as in this case every entry of $W(0)$ must be 1 since it considers every possible length of the waiting-time process for each final phase of the time-reversed service process. We therefore find the diagonal matrix Γ such that

$$\Gamma W(0) = \Gamma(T-S)'\Lambda\mathbf{1} = \mathbf{1}.$$

Notice that, from Eq. (2.9), $T - S$ is a non-negative matrix since T has non-zero diagonal elements [Sen89]. Thus, from the definition of Γ, the matrix $\tilde{P}_{s,w} = \Gamma^{-1}(T-S)'\Lambda$ is a stochastic matrix. This leads to the second and third blocks in Eq. (3.15), which show how, for a job that must wait, the response-time process starts with the time-reversed service process, with parameters $(\tilde{\boldsymbol{\alpha}}_{\mathrm{service}}, \tilde{S}_{\mathrm{service}})$. After this, the waiting-time process starts, selecting the initial

phase of the time-reversed waiting process according to the matrix $\tilde{P}_{s,w}$. The process then continues according to the sub-generator S_{wait} until absorption. □

A.1.3 Proof of Theorem 5.1

Proof. To build the PH representation of the response-time distribution we consider separately the paths of jobs that start service immediately after arrival from those that must wait. As stated before, a job starts service without waiting with probability $1-\gamma$, its initial service phase is given by $\boldsymbol{\pi}(0)$, and the sub-generator associated to its service-time is S_{ser}. This is reflected in the first set of phases in Eq. (5.2).

If a job has to wait, its response time is composed of a first stage of waiting followed by a second stage of service. Further, the phase in which the waiting stage ends determines the stage in which the service stage begins. Thus we need to keep track of the phase in which the waiting-time concludes, and find the transition probabilities with which the service stage starts given the phase in which the waiting stage ended. However, the PH representation of the waiting-time distribution cannot be used directly for this purpose as it results from a time-reversal argument [Sen89]. To overcome this, we find the time-reversed version of the service-time distribution, and determine the probability that the time-reversed waiting-time starts in each phase, given the final phase of the time-reversed service-time stage.

The PH representation of the time-reversed service-time distribution $(\boldsymbol{s}_{\text{ser}}, S_{\text{ser}})$ is given by [Ram90]

$$\tilde{\boldsymbol{s}}_{\text{ser}} = \Delta^{-1}\boldsymbol{s}'_{\text{ser}}\Delta, \quad \tilde{\boldsymbol{\alpha}}_{\text{busy}} = (-\boldsymbol{s}_{\text{ser}}\mathbf{1})'\Delta,$$

where Δ is a diagonal matrix such that $\Delta\mathbf{1} = \boldsymbol{\eta}'$, and $\boldsymbol{\eta} = -\boldsymbol{s}_{\text{ser}}S_{\text{ser}}^{-1}$ is the stationary distribution of the phase during service. This distribution is such that its initial phase is actually given by the final phase of the original distribution, and vice versa.

To find how the phase transitions from the end of the service stage to the beginning of the waiting stage, in the time-reversed process, we observe that the joint probability that after a

downward jump the level is at least x and the phase is j is given by the j^{th} entry of

$$W(x) = \int_x^\infty \boldsymbol{\pi}(y) \int_0^\infty \exp(Tu) A^{(\text{MAP})}(u) du dy,$$

as this expression considers the probability that any level greater than x is visited by means of a downward jump of any size u. From (2.11) we know that the inner integral is equal to $T - S^{(\text{MAP})}$, obtaining

$$\begin{aligned} W(x) &= \int_x^\infty \boldsymbol{\pi}(y)(T - S^{(\text{MAP})}) dy = -\boldsymbol{\pi}(0) T^{-1} \exp(Tx)(T - S^{(\text{MAP})}) \\ &= \boldsymbol{\alpha}_{\text{busy}} \exp(Tx)(T - S^{(\text{MAP})}). \end{aligned}$$

Now we perform the same similarity transform as in [Sen89] to obtain the PH representation of the waiting-time process. We thus define a diagonal matrix Λ such that $\Lambda \mathbf{1} = \boldsymbol{\alpha}'_{\text{busy}}$ to obtain

$$\begin{aligned} W(x) &= \boldsymbol{\alpha}_{\text{busy}} \Lambda^{-1} \Lambda \exp(Tx) \Lambda^{-1} \Lambda (T - S^{(\text{MAP})}) = \mathbf{1}' \exp(S'_{\text{wait}} x) \Lambda (T - S^{(\text{MAP})}) \\ &= (T - S^{(\text{MAP})})' \Lambda \exp(S_{\text{wait}} x) \mathbf{1}. \end{aligned}$$

We therefore have the standard representation for the sub-generator of the waiting-time distribution, but the term $(T - S^{(\text{MAP})})' \Lambda$ captures the transition from the final phase in the time-reversed service process to the initial phase in the time-reversed waiting-time process. We now normalize this expression by evaluating $W(x)$ at $x = 0$, as in this case every entry of $W(0)$ must be 1 since it considers every possible length of the waiting-time process for each final phase of the time-reversed service process. We therefore find the diagonal matrix Γ such that

$$\Gamma W(0) = \Gamma (T - S^{(\text{MAP})})' \Lambda \mathbf{1} = \mathbf{1}.$$

Notice that, from (2.11), $T - S^{(\text{MAP})}$ is a non-negative matrix since T has non-zero diagonal elements [Sen89]. Thus, from the definition of Γ, the matrix $\tilde{P}_{s,w} = \Gamma^{-1}(T - S)' \Lambda$ is a stochastic matrix. This leads to the second and third blocks in Eq. (5.2), where the response-time process starts with the time-reversed service process, with parameters $(\tilde{\boldsymbol{\alpha}}_{\text{busy}}, \tilde{S}_{\text{ser}})$. After the service completes, the waiting-time process starts, selecting the initial phase of the time-reversed wait-

Algorithm 2 EM algorithm for the CHE service model

Require: Data: $\mathcal{D}$, Number of phases: m_R, ϵ
1: $\mathcal{S} = \{(B, h_1, \ldots, h_B) | \sum_{b=1}^{B} h_i = m_R\}$
2: bestLH $= \infty$
3: **for** $(B, h_1, \ldots, h_B) \in \mathcal{S}$ **do**
4: Initial guess $\hat{\Theta} = (\hat{\alpha}_1, \ldots, \hat{\alpha}_B, \hat{\lambda}_1, \ldots, \hat{\lambda}_B)$
5: diff $= 1$
6: **while** diff $> \epsilon$ **do**
7: Compute $\delta(b|x_n, \hat{\Theta})$, $b = 1, \ldots, B$, $d = 1, \ldots, D$ as in (A.4), (A.2)
8: Compute α_b, λ_b, $b = 1, \ldots, B$ as in (A.1)
9: diff $= \max |\hat{\Theta} - (\alpha_1, \ldots, \alpha_B, \lambda_1, \ldots, \lambda_B)|$
10: $\hat{\Theta} = (\alpha_1, \ldots, \alpha_B, \lambda_1, \ldots, \lambda_B)$
11: **end while**
12: Compute $\log L(\hat{\Theta}|\mathcal{D})$ as in (A.3)
13: **if** $\log L(\hat{\Theta}|\mathcal{D}) >$ bestLH **then**
14: $\Theta^* = \hat{\Theta}$
15: bestLH $= \log L(\hat{\Theta}|\mathcal{D})$
16: **end if**
17: **end for**
18: **return** Θ^*

ing process according to the matrix $P_{s,w}$. Notice that absorption from the service-time process can only occur in its first m phases, which correspond to the *full* set of phases defined in Section 4.2.2. The matrix $\tilde{P}_{s,w}$ corresponds to this m phases. The process then continues according to the sub-generator S_{wait} until absorption. □

A.1.4 Fitting correlated processing times

The CHE service-time model introduced in Section 5.2.5 allows the r replicas of a job to be correlated in the sense that all of them select the same Erlang branch of the CHE distribution. The benefits of the CHE service model can only be exploited as long as we are able to fit the model parameters from a data trace. While fitting methods exists for general PH distributions [ANO96] and some of its sub-classes [KSH03, TBT06], no such method exist for the CHE service model since this is the first time this model is proposed. In this section, we propose a maximum-likelihood method for this purpose, based on the Expectation Maximization (EM) algorithm proposed by Dempster et al. [DLR77]. Because our analysis method obtains the response-time *distribution*, its ability to provide accurate results depends on capturing the processing-time *distribution* well. As a maximum-likelihood approach considers the overall

processing-time distribution, it appears better suited than the other widely adopted class of fitting method, moment matching [Whi82], which captures only the first few moments. Also, the popularity of the EM method for PH distributions is due to its ability to handle mixtures of distributions, which we can also exploit for CHE distributions.

We assume a trace $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_D\}$, where each of the D observations is a tuple $\boldsymbol{x}_d = (x_d^1, \ldots, x_d^r)$ holding the processing times of the r replicas of a job. From this trace, we want to determine the entire set of parameters of the CHE model $(B, h_1, \ldots, h_B, \alpha_1, \ldots, \alpha_B, \lambda_1, \ldots, \lambda_B)$. However, and similar to [TBT06], we split this problem in two. First, we assume the number of branches B, and the number of phases h_i in each branch is known. Thus, we focus on determining the parameters $\Theta = (\alpha_1, \ldots, \alpha_B, \lambda_1, \ldots, \lambda_B)$, that is, the probability α_i of choosing each branch and the corresponding Erlang rate λ_i, for $i \in \mathcal{B}$. Later in this section, we consider how to determine B and h_i.

The basic idea behind the EM algorithm is to start with an initial guess of the parameters $\hat{\Theta}$. We then iterate to obtain new values of $\hat{\Theta}$ that maximize the log-likelihood function by following the derivation detailed in Appendix A.2. Specifically, we compute $\delta(b|\boldsymbol{x}_d, \hat{\Theta})$, the probability that branch b is selected given data $\boldsymbol{x}_d$, and estimates $\hat{\Theta}$, which is the expectation step in the EM method. In the optimization step, we obtain the branch selection probabilities and the Erlang rates as

$$\begin{aligned} \alpha_b &= \frac{1}{D} \sum_{d=1}^{D} \delta(b|\boldsymbol{x}_d, \hat{\Theta}), \\ \lambda_b &= \frac{r h_b \sum_{d=1}^{D} \delta(b|\boldsymbol{x}_d, \hat{\Theta})}{\sum_{d=1}^{D} \delta(b|\boldsymbol{x}_d, \hat{\Theta}) \sum_{j=1}^{r} x_d^j}. \end{aligned} \tag{A.1}$$

As evidenced in this last expression, derived in Appendix A.2, the key characteristic of this method, and what makes it different from existing ones, is that it considers each sample $\boldsymbol{x}_d$ as an r-tuple, and these r processing-times samples are tied by the selection of the same Erlang branch, in agreement with the CHE model.

We are now ready to state the EM algorithm for the CHE service model, summarized in Algorithm 2. The algorithm requires the total number of phases m_R, the data $\mathcal{D}$, and a stopping criterion ϵ. As m_R is equal to the sum of the number of stages in all branches, $\sum_{i \in \mathcal{B}} h_i$, the

algorithm determines all possible combinations for the number of branches and number of stages in each branch $(B, h_1, \ldots, h_B)$, as in [TBT06] for the independent hyper-Erlang case. For each possible combination, it then executes the EM steps derived in Appendix A.2, starting from an initial guess for the parameters and iterating until the difference between two successive sets of estimates in Eq. (A.1) is less than the pre-defined limit ϵ. Once the estimates have been found, it computes the log-likelihood and compares it against the best one found so far, keeping the set of estimates Θ^* with the highest log-likelihood, which is returned at the end of execution. Note that considering all possible combinations $(B, h_1, \ldots, h_B)$ for a given number of phases m_R is feasible as long as this number is moderate. Given that the CHE service model is to be used for analysis as described in Section 5.2.5, we are actually interested in small to moderate values for m_R, and the full enumeration is thus feasible. In Section 5.2.6, we illustrate how this method is able to capture the correlation structure that emerges in the processing times of request replicas from a real system.

A.2 EM estimation for CHE times

The key to the EM algorithm is to consider the dataset $\mathcal{D}$ as incomplete data, where the branch z_d selected in sample d is missing. If this information, summarized in the vector $\boldsymbol{z}$ of size D, was known we could write the log-likelihood of the parameter set Θ as

$$\log L(\Theta|\mathcal{D}, \boldsymbol{z}) = \sum_{d=1}^{D} \log\left(\alpha_{z_d} p_{z_d}(\boldsymbol{x}_d|\lambda_{z_d})\right),$$

where $p_b(\boldsymbol{x}_d|\lambda_b)$ is the likelihood of observing the response times $\boldsymbol{x}_d$ when branch b is selected and has rate λ_b. Given the CHE service model, this quantity is

$$p_b(\boldsymbol{x}_d|\lambda_b) = \frac{\left(\lambda_b^r \prod_{j=1}^{r} x_d^j\right)^{h_b-1}}{((h_b-1)!)^r} \lambda_b^r \exp\left\{-\lambda_b \sum_{j=1}^{r} x_d^j\right\}, \tag{A.2}$$

because, given that branch b has been selected for sample d, the processing times of the r replicas follow independent Erlang distributions with parameters (h_b, λ_b). Note that the log-likelihood

without considering $\boldsymbol{z}$ is given by

$$\log L(\Theta|\mathcal{D}) = \sum_{d=1}^{D} \log \left(\sum_{b=1}^{B} \alpha_b p_b(\boldsymbol{x}_d|\lambda_b) \right), \tag{A.3}$$

which considers all possible branches that could be selected in each sample.

As the vector $\boldsymbol{z}$ is in fact a realization of a random variable Z, the next step is to assume that a guess $\hat{\Theta}$ of the parameters is known. With this, we can determine the expected value of the log-likelihood with respect to the random variable Z given the observations $\mathcal{D}$ and the guessed parameters $\hat{\Theta}$, which we write as $Q(\Theta, \hat{\Theta}) = E_Z \left[\log L(\Theta|\mathcal{D}, \boldsymbol{z}) | \mathcal{D}, \hat{\Theta} \right]$. Since Z lives in the space $\zeta = (1, \ldots, B)^D$, we can write this expected value as

$$Q(\Theta, \hat{\Theta}) = \sum_{\boldsymbol{z} \in \zeta} \sum_{d=1}^{D} \log \left(\alpha_{z_d} p_{z_d}(\boldsymbol{x}_d|\lambda_{z_d}) \right) \prod_{i=1}^{D} \delta(z_i|\boldsymbol{x}_i, \hat{\Theta}),$$

where the first term in the sum is the likelihood of observing the processing time trace $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_D)$ given that the branches selected were $\boldsymbol{z}$, and the second term holds the probability that branches were selected as in $\boldsymbol{z}$. If we consider each possible branch selected in each sample, we can re-write this expression as

$$Q(\Theta, \hat{\Theta}) = \sum_{d=1}^{D} \sum_{b=1}^{B} \log \left(\alpha_b p_b(x_d|\lambda_b) \right) \delta(b|\boldsymbol{x}_d, \hat{\Theta}).$$

A detailed derivation of this expected value for the hyper-Erlang case can be found in [TBT06, Appendix A.2]. In fact, we rely on [TBT06] to find $\delta(b|\boldsymbol{x}_n, \hat{\Theta})$ by means of Bayes' rule as

$$\delta(b|\boldsymbol{x}_d, \hat{\Theta}) = \frac{\delta(b|\hat{\Theta}) p(\boldsymbol{x}_d|b, \hat{\Theta})}{p(\boldsymbol{x}_d|\hat{\Theta})} = \frac{\hat{\alpha}_b p_b(\boldsymbol{x}_d|\hat{\lambda}_b)}{\sum_{j=1}^{B} \hat{\alpha}_j p_j(\boldsymbol{x}_d|\hat{\lambda}_j)}, \tag{A.4}$$

which gives us the likelihood of choosing branch b for sample d as a ratio that compares it with all branches that could be selected. Computing these quantities constitutes the expectation step of the EM method.

For the maximization step we find the values of Θ that maximize $Q(\Theta, \hat{\Theta})$. As the CHE service model is a mixture of densities, it has been shown in [DLR77] that we can obtain α_b as in

Eq. (A.1). To obtain λ_b we derivate $Q(\Theta, \hat{\Theta})$ with respect to λ_b,

$$\frac{\partial Q}{\partial \lambda_b} = \sum_{d=1}^{D} \delta(b|\boldsymbol{x}_d, \hat{\Theta}) \frac{\partial}{\partial \lambda_b} \log \left(\frac{\left(\lambda_b^r \prod_{j=1}^{r} x_d^j\right)^{h_b - 1}}{((h_b - 1)!)^r} \lambda_b^r \exp\left\{ -\lambda_b \sum_{j=1}^{r} x_d^j \right\} \right)$$

and equate to zero to obtain Eq. (A.1).