Response Time Distribution of Flash Memory Accesses

Peter G. Harrison[1] Naresh M. Patel[2] and Soraya Zertal[3]

[1] Imperial College London, South Kensington Campus, London SW7 2AZ, UK
pgh@doc.ic.ac.uk

[2] NetApp, Inc., 495 East Java Drive, Sunnyvale, CA 94089, USA
naresh@netapp.com

[3] PRiSM, Université de Versailles, 45, Av. des Etats-Unis, 78000 Versailles, France
Zertal@prism.uvsq.fr

Abstract

Flash memory is becoming an increasingly important storage component among non-volatile storage devices. Its cost is decreasing dramatically and its performance continues to improve, which makes it a serious competitor for disks and a candidate for enterprise-tier storage devices of the future. Consequently, it is important to devise models and tools to analyse its behaviour and to evaluate its effects on a system's performance. We propose a Markov modulated fluid model with priority classes to investigate the response time characteristics of Flash memory accesses. This model can represent well the Flash access operation types, respecting the erase/write/read relative priorities and autocorrelations. We apply the model to estimate response time densities at the chip for an OLTP-type of workload and indicate the magnitude of the penalty suffered by writes under priority scheduling of read operations. The model is validated against a customised hardware simulator that uses input-traces typical of our Markovian workload description.

keywords: Fluid model, Flash memory, Response time distribution, Preemptive priority.

1 Introduction

Storage devices based on Flash memory are becoming more and more prevalent in our daily life. This recent technology presents a panoply of devices, subject to continuous and intensive evolution – as commonly observed in the semiconductor manufacturing industry but especially prominent nowadays in response to the market demand of mp3 players, mobile phones, digital cameras, solid state drives and other consumer and enterprise storage products.

However, there are currently very few tools capable of analysing the technology's quantitative behaviour and its effect on the whole system of which it forms a part. Moreover, there are no methodologies available to evaluate the performance it can deliver and the associated cost of its very specific operating characteristics. Such tools and methodologies must be representative of the system under scrutiny, accurate at the scale of the semiconductor and quick to develop and use, in order to remain in step with the wide variety of devices available and their rate of

manufacture. The mathematical modelling techniques we present address these requirements with respect to both their ease of representation and accuracy. This requires careful model design, parameterisation, implementation and validation.

A considerable body of work has been done on the usage/access (for example [3, 4, 13]) and reliability (such as in [21, 23, 24]) aspects of Flash, focussing on specific file systems [17, 20, 16, 14, 12, 15, 22]. On the other hand, there is currently no analytical model of Flash memory performance to the authors' best knowledge. The closest approach, a project that is still in progress, is that of Butterfield *et al.* [1, 2], in which fragments of Z-models of NAND Flash are considered. This work is based on the "open NAND Flash Interface (ONFI)" consortium [6] and aims to give a formal, functional description of NAND Flash memory without any evaluation of its performance or analysis of its behaviour when submitted to different types and intensities of workload.

We propose a fluid model for a NAND Flash memory. Fluid models represent the behaviour of Flash memory well because they can take into account correlation between and autocorrelation within access streams – of erase, write, and read operations – as well as their relative priorities. As in [5], we consider three different types of requests and a Flash component with an intermediate (fluid) buffer. A Markov chain is used to describe the current mode of the offered access operations, in particular their type and input rate. Although the tools supporting fluid models are not yet as well developed as those for discrete state Markov models [8], there is a wide range of applications that can be modelled effectively using them, and cannot be modelled easily using discrete methods; Flash memory access is one such.

The next section of this paper describes the background to Flash technology – its physical characteristics, its access operations and its impact in the entreprise storage system domain. Section 3 describes the fluid model and how it is applied to represent our particular Flash memory system. We analyse both the low (write / erase) and high (read) priority storage access classes, as well as the corresponding no-priority system, in terms of their respective response times, using busy period analysis. As a case study, we consider an online transaction processing (OLTP) system in section 4, which clarifies how the proposed, generic fluid model can be customised to represent a particular Flash storage system supporting this specific type of workload. Numerical results are given in section 5, comprising graphs of queueing (or response) time probability densities together with plots of mean queueing time against chip utilisation. The latter plots are compared with corresponding estimates from a customised hardware simulator. Finally, we conclude in section 6, suggesting model enhancements, further directions for analysis and new designs for this recent technology that may emerge in response to our modelling methodology.

2 Background on Flash memory

The impact of Flash memory as a major, non-volatile storage component stems from its shock resistance, vibration tolerance, light weight and low energy consumption, as well as its high capacity. It already has a wide range of applications, from its use in personal computers as a solid-state drive (SSD) to critical environments such as satellite systems [13, 24]; in addition to the consumer product applications referred to in the Introduction.

There are two types of Flash memory, named according to their manufacturing components: NOR and NAND Flash. The former has lower density and higher cost but provides fast random access and can be easily re-programmed; this makes it most suitable for storing codes. Another advantage of NOR is its lower susceptibility to corruption than NAND, partly because of the bad blocks that exist in the latter from the time of manufacture.

On the other hand, NAND Flash has a very large storage capacity and provides high performance for large read/write requests, making it more suitable for storing data [14]. In fact the density is about 8 times more for NAND, at a cost that is 4 to 8 times cheaper than NOR [17]. Further, sequential read and write accesses are faster on NAND Flash. Although erases are significantly faster on NOR, they can be pre-scheduled in NAND, essentially running in a garbage collector. Finally, MLC_n (Multi Level Cell) technology multiplies the storage capacity of a Flash memory chip by having n-bit information per cell, where $n = 2$ or 4 in contemporary devices.

All of the previous characteristics make NAND the technology of choice for many data centres and this is what we model here, using fluid methods. We model essentially the basic SLC (Single Level Cell) technology with only one bit per cell, i.e. effectively MLC_1, but it is straightforward to adapt the techniques used to arbitrary n in the mapping from discrete storage requests onto volumes of fluid – see section 4.

2.1 NAND Flash memory access

A NAND Flash memory chip is composed of a fixed number of blocks, each of which is partitioned into a fixed number of pages. Every page consists of two areas: a data area for native (user) data and a spare area for data status information (figure 1). A block is the erase operation's unit of storage, whilst a page is the

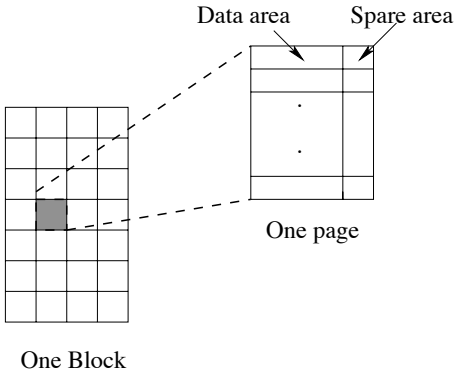

Figure 1: A NAND Flash block structure

read and write operation's unit. No 'in-place' updates are allowed in NAND Flash and pages have to be written in order within a block. So, when data is modified, the new version must be written to an available page – called the *live page*. The page containing the old version is considered a dead page and is invalidated. As time passes, the number of dead pages increases and the system reclaims them, in order to perform further write operations, by running a garbage collection process. This is possible only when these pages have been erased. However, this erase/write unit mismatch generates additional copying of live pages from one block, when erasing it, to another block. Another limitation of the NAND Flash technology is that the number of erase operations is limited to about 10^5 for SLC and 10^4 for MLC_2 [4, 23]. As any recycling of dead pages introduces block erasing, an even erase-count distribution over the Flash memory blocks cannot be achieved, which results in the "wear-levelling" problem. This has a significant negative impact on the longevity of the memory chip.

Much like the small write problem in traditional RAID5 systems, random writes to a Flash device require a read-modify-write sequence to overcome the lack of in-place updates: read the old block, modify updated pages, and write the new block.

Some form of mapping function is needed to hide this from the user and the next subsection outlines some common approaches.

2.2 Flash management systems

Several file systems have been developed to manage data on Flash memories. JFFS (Journal Flash File System) is a log-structured file system for the NOR Flash device [20]. Its second version (JFFS2) supports NAND devices with a sequential I/O interface and a more efficient garbage collection process, using a relatively reduced mounting time. YAFFS (Yet Another Flash File System) is the first file system designed specifically for NAND devices, considering data integrity as a priority [15], and its second version (YAFFS 2) accommodates a newer chip with larger pages. More recently, LogFS [16] supports snapshots and is better suited to large devices due to its reduced mounting time and its efficient garbage collection process [12]. Finally, we cite [22] for a review of hybrid architectures incorporating both Flash and RAM.

All of these Flash file systems have an FTL (Flash Translation Layer) composed essentially of two parts: an allocator process for the logical to physical address space mapping and a cleaner process for garbage collection. The mapping between the logical location and the physical location is performed using metadata in the pages' spare areas, mounted at the initialisation phase before any I/O operation has taken place. Garbage collection is performed in the background to make free space for write operations. The block size is an important design parameter which has a significant impact on the amount of metadata required and so mounting time. The flash controller therefore has many design choices, which affect the output stream of storage requests it delivers to the Flash devices. Hence we consider a fairly general input stream, using a Markov modulated arrival process to model the input to a Flash memory array. More specific models, tailored to particular Flash controller designs, could be developed subsequently by appropriately parameterising the base model we describe here.

2.3 Flash for enterprise storage

As more consumer devices incorporate Flash and new process technology is implemented, the cost of Flash is expected to decrease rapidly in the coming years. This trend may allow Flash to compete with high-speed enterprise disks in the future. Most enterprise storage systems already provide fast writes using Non-Volatile RAM but random read operations from disk incur long latencies (typically 5 to 10 milliseconds). On the other hand, sequential read operations from disk are very fast and so the randomness of the access pattern significantly impacts the observed performance. The key advantage of Flash is that the random reads and sequential reads have the same low latency. On the downside, Flash implementations have to deal with the complexities of the write constraints outlined above.

While 4KB page read operations from Flash have fairly low average latency (typically around 25 microseconds for access and 105 microseconds for channel transfer), the distribution of the latency may have a long tail because of contention with write and erase operations. Page write operations take about 105 microseconds on the channel and 200 to 700 microseconds for access (often an increasing function of erase count). Erasing a block (say 256 KB, or 64 pages) can take more than 1.5 milliseconds. These numbers vary considerably among the Flash device manufacturers, but often the latency concern stems from the possibility that small read operations have to wait behind relatively long write and erase operations. Hence we would like to model the distribution of latency for read operations (in particular) under various workloads. Fluid modelling techniques provide a promising way of improving our

intuition about response time distributions for given Flash device characteristics, as well as providing quantitative estimates.

3 Fluid model

We use a fluid queue, with input defined by a four-state (or four-phase) continuous time semi-Markov chain (CTSMC), to model the behaviour of a Flash device. Such a queue is appropriate to model correlated streams of incoming data requests of different types, especially (but not exclusively) at moderate to high utilisation. In fact, in high utilisation scenarios, the erase operations and their related page copying can be generated frequently due to the high rate of native write/update operations. The four states, numbered 0 to 3, correspond to *off* (no input), *read requests*, *write requests* and *erases*. Class 1 fluid 'particles' (i.e. those arriving in state 1) have priority over states 2 and 3 to account for the priority actually given to reads, which are usually more critical in allowing a process to continue.

The queue is parameterised, for an n-state CTSMC so as to allow for model extensions with more than four states, as follows:

- the CTSMC has probability transition matrix $P = (p_{ij} \mid 0 \leq i, j \leq n-1)$, defined at state transition instants;
- the state holding time in state i, given that the next state is j, has probability distribution $H_{ij}(t)$, for $0 \leq i, j \leq n-1$.
- the fluid arrival rate in state i is the constant λ_i volume-units of fluid per unit time, for $0 \leq i \leq n-1$, so $\lambda_0 = 0$;
- the rate at which the server outputs fluid when its buffer is non-empty is the constant μ volume-units of fluid per unit time;
- the net input *rate matrix* $R = \mathrm{diag}(r_0, \ldots, r_{n-1})$, where $r_i = \lambda_i - \mu$ for $0 \leq i \leq n-1$, and the *rate vector* $\vec{r} = (r_0, \ldots, r_{n-1})$;
- the vector $e = (1, 1, \ldots, 1)$.

In fact we will require that all states other than the high-prority state 1 have exponential holding times with parameter that does not depend on the next state transited to; these are characterised by a *rate* γ_i for state $i = 0, 2, 3$, so that $H_{ij}(t) = 1 - e^{-\gamma_i t}$ for all states $j \neq i$. In this paper, we further assume that state 1 also has exponential holding time, so we have a modulating continuous time Markov chain (CTMC), specified by a generator matrix, Q say[1].

The response time distribution (or, rather, its Laplace-Stieltjes transform (LST)) of each class in this fluid queue can be calculated using the following observations:

1. high priority fluid (arriving in state 1) is processed as if the other classes had fluid arrival rates 0, which effectively constitute additional off-states.
2. response time for a state-1 particle is therefore the time taken to serve the state-1 fluid in the buffer present at the arrival instant of that particle – that is, simply the volume of state-1 fluid present divided by μ, given by the reduced model described in the previous point.

[1]This assumption is not necessary, however, since it is only used to determine the equilibrium fluid level distribution, which can be derived fairly simply as a functional equation for an on-off process in which the off-period is exponential.

3. response time for particles arriving in lower priority states is the sum of the time to serve all fluid (of both classes) already present on arrival – of volume F say – added to all additional fluid of class 1 that arrives during that time period. The amount of this additional fluid is the product of λ_1 and the sum of the *busy periods* generated by each of these state-1 arrivals. When there is only one off-state (i.e. in an on-off process), the number of these busy periods, when conditioned on the time period F/μ, is a Poisson random variable. However, this is not the case here and a more complex analysis is given in section 3.4. Note that the fluid queue model for the period considered is that described in the first point since lower class fluid waits behind class 2 and so can be ignored here. However, F is given by the steady state fluid level in the original 4-state fluid queue *with no priority classes*, because of the service rate being the same for all classes. The probability distribution of this fluid level is well known for any number of states with any fluid input rates.

We therefore have to calculate:

- the steady state probability distribution of the fluid level in the no-priority queue (for the low priority class) and in the high priority class only queue (for the high priority class) – a routine task;
- the busy period for a class 1 fluid arrival in the reduced model – this is determined in section 3.2 using results from [9];
- the LST of the queueing time[2] distribution for each class – this is done in section 3.4 in terms of the time required to clear F units of fluid from the buffer.

3.1 Non-priority fluid queue results

Here we restate relevant results for a single fluid queue with a Markov modulated arrival process of any number of states. Consider a single, n-state Markovian fluid queue, with generator matrix $Q = (q_{ij} \mid 1 \leq i, j \leq n)$ for the input process, which has equilibrium probabilities $\vec{\pi}$ (so that $\pi Q = \vec{0}$ and $\pi \vec{e}^{\mathrm{T}} = 1$).

Using the notation defined in the previous section, it can be shown that the vector density function of the fluid level at equilibrium (when this exists), denoted by $\vec{f}(x) = \frac{\partial \vec{F}}{\partial x}$ where $\vec{F}(x)$ is the corresponding probability distribution function, has Laplace transform:

$$\vec{f}^*(\theta) = \vec{F}(0)R(R - Q/\theta)^{-1} \tag{1}$$

The constant $\vec{F}(0)$ is given by the boundary conditions that the fluid level in states with fluid arrival rate greater than the service rate is positive with probability 1, and taking the limit $\theta \to 0$. Details can be found in [7], but note that this is not the only solution method; see for example [5].

For the high priority class's response time, we can consider the queue as if the other classes had zero arrival rate. To get the equilibrium probability density function of the volume of class 1 fluid in the queue, we simply solve the same problem with $\lambda_2 = \lambda_3 = 0$. Let this density function have Laplace transform denoted by $\vec{f_1}^*(\theta)$, that for the non-priority fluid level F being $\vec{f}^*(\theta)$, as above.

3.2 High priority class busy times

The model we solve here again has four states, but three of these have zero fluid arrival rate at any given time t. Consistent with [9], the set of "emptying-states"

[2]Since a single fluid particle is served in negligible time, its response time and queueing time are essentially the same.

(or "draining-states", in which the fluid level decreases) is denoted by $\mathcal{E} = \{0, 2, 3\}$ and the set of all states by $\mathcal{S} = \{0, 1, 2, 3\}$. The number of draining-states is $n_e = |\mathcal{E}|$ and the total number of states is $n = |\mathcal{S}|$; thus in our example, $n_e = 3$ and $n = 4$. Compared to the case where $n_e = 2$, implemented in [9], we need to solve a cubic equation for the parameter β (below), rather than a quadratic, and then three simultaneous quadratic equations, rather than two. The programming and computational effort is therefore considerably greater; we used Mathematica® Version 5.2 to do this.

An outline of the method we use is as follows. We first define the following matrices:

- the $n \times n$ matrix $\Gamma = (\gamma_{ij})$, the generator matrix of the modulating n-state Markov process, where we write $\gamma_i = -\gamma_{ii}$ for the positive total transition rate out of state $i \in \mathcal{S}$.
- the $1 \times n_e$ matrix (vector) $\mathbf{V} = (V_{1j}^*(\alpha))$, the j element of which is the Laplace transform, with real parameter $\alpha \geq 0$, of the probability density function of the busy period – i.e. of the time elapsed between entering into the single filling-state 1 (for reads) when the queue is empty (hence, coming from a draining-state) and the queue next becoming empty again in the draining-state $j \in \mathcal{E}$.
- the $n_e \times n$ matrix $\mathbf{U} = (u_{ij})$ by $u_{ij} = -\gamma_{ij}(1 - \delta_{ij})/r_i$ for $i \in \mathcal{E}$ and $j \in \mathcal{S}$.
- the $n_e \times n_e$ diagonal matrix $\mathbf{D} = (d_{ij})$ by $d_{ij} = [\beta - (\alpha + \gamma_i)/r_i]\delta_{ij}$ for $i, j \in \mathcal{E}$, where $\beta \in \mathbb{C}$.
- the $n \times n_e$ matrix $\mathbf{W} = (w_{ij})$ by $w_{1j} = V_{1j}^*(\alpha)$, $w_{ii} = 1$ for $i \in \mathcal{E}$ and $w_{ij} = 0$ otherwise, for $j \neq i \in \mathcal{E}$.

The $n_e \times n_e$ matrix $\mathbf{M} = \mathbf{D} - \mathbf{UW}$ has determinant, $\Delta \stackrel{\text{def}}{=} |\mathbf{M}|$ say, which is a polynomial of degree n_e in β and whose cofactors are polynomials of degree $n_e - 1$. Therefore each element in the inverse matrix $(\mathbf{D} - \mathbf{UW})^{-1}$ (assuming this exists, i.e. that the determinant $|\mathbf{M}|$ is non-zero) is a rational function of β. We write this in partial fractions as

$$\mathbf{L} = (\mathbf{D} - \mathbf{UW})^{-1} = \left(\sum_{s=1}^{n_e} \frac{a_{s;ij}(\alpha)}{\beta - b_s(\alpha)} \mid i, j \in \mathcal{E} \right)$$

assuming no degeneracies, where $\{b_s(\alpha) \mid 1 \leq s \leq n_e\}$ are the distinct roots of the equation $\Delta = 0$ (in β) and the $a_{s;ij}(\alpha)$ are independent of β, defined by (dropping the parameter α when the meaning is clear):

$$a_{s;ij} = \left[\frac{c_{ji}}{\Delta_s} \right]_{\beta = b_s} \tag{2}$$

where $\mathbf{C} = (c_{ij} \mid i, j \in \mathcal{E})$ is the cofactor matrix of $\mathbf{M}$ and $\Delta_s = \Delta/(\beta - b_s) \equiv \prod_{1 \leq i \neq s \leq n_e} (\beta - b_i)$. Our special case of only a single filling-state (or on-state) now gives a simplified result for the Laplace transform of the high priority class busy time probability distribution, V_{1j}^*, where j denotes the (draining) state in which the busy period ends [9]:

Theorem 1 *For $j \in \mathcal{E} = \{0, 2, 3\}$,*

$$V_{1j}^*(\alpha) = \sum_{k \in \mathcal{E}} \sum_{s \in \mathcal{E}} a_{s;kj}(\alpha) p_{1k} H_{1k}^*(\alpha - r_1 b_s(\alpha))$$

for general holding times in the high priority state 1. In the Markovian case (when the holding time in state 1 is exponential with parameter γ_1)

$$V^*_{1j}(\alpha) = \sum_{k\in\mathcal{E}}\sum_{s\in\mathcal{E}} \frac{a_{s;kj}(\alpha)q_{1k}}{\gamma_1+\alpha-r_1 b_s(\alpha)}$$

Notice that the terms $a_{s;kj}(\alpha)$ are functions of the busy period Laplace transforms $V^*_{1i}(\alpha)$ $(i,j,k \in \mathcal{E})$, giving a set of mutual, non-linear equations. Of course, if $n_e = 1$, there is only one such equation, for $V^*_{10}(\alpha)$, which is the well known one for the busy period in a fluid queue with on-off source.

We will also require the related quantity $Z^*_{ij}(x,\alpha)$ for each $i,j \in \mathcal{E}, x > 0$, which is the LST of the probability distribution of the time taken, $Z_{ij}(x)$, for a buffer containing x units of fluid in draining state i to first become empty in state j. The following result gives this:

Theorem 2 *For $i,j \in \mathcal{E}$ and real $x > 0$,*

$$Z^*_{ij}(x,\alpha) = \sum_{s=1}^{n_e} a_{s;ij}(\alpha)e^{b_s(\alpha)x}$$

Theorems 1 and 2 have been implemented in Mathematica® and our numerical results are obtained using them, together with the analysis of queueing times given in the next section. To illustrate the theorems' use, we first need the matrix $\mathbf{M}$ to calculate the terms b_s and $a_{s;kj}$, for $j,k,s = 0,2,3$.

The matrix $\mathbf{D} = \mathrm{diag}(\beta + (\alpha+\gamma_0)/\mu, \beta + (\alpha+\gamma_2)/\mu, \beta + (\alpha+\gamma_3)/\mu)$, $u_{ij} = \gamma_{ij}/\mu$ $(i = 0,2,3; j = 0,1,2,3)$ and

$$\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 \\ V_{10}(\alpha) & V_{12}(\alpha) & V_{13}(\alpha) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Hence

$$\begin{aligned} \mathbf{M} &= \mathbf{D} - \mathbf{UW} \\ &= \begin{pmatrix} \beta + \frac{\alpha+\gamma_0-\gamma_{01}V^*_{10}(\alpha)}{\mu} & -\frac{\gamma_{02}+\gamma_{01}V^*_{12}(\alpha)}{\mu} & -\frac{\gamma_{03}+\gamma_{01}V^*_{13}(\alpha)}{\mu} \\ -\frac{\gamma_{20}+\gamma_{21}V^*_{10}(\alpha)}{\mu} & \beta + \frac{\alpha+\gamma_2-\gamma_{21}V^*_{12}(\alpha)}{\mu} & -\frac{\gamma_{23}+\gamma_{21}V^*_{13}(\alpha)}{\mu} \\ -\frac{\gamma_{30}+\gamma_{31}V^*_{10}(\alpha)}{\mu} & -\frac{\gamma_{32}+\gamma_{31}V^*_{12}(\alpha)}{\mu} & \beta + \frac{\alpha+\gamma_3-\gamma_{31}V^*_{13}(\alpha)}{\mu} \end{pmatrix} \end{aligned}$$

$b_1(\alpha), b_2(\alpha), b_3(\alpha)$ are the roots (assumed distinct) of the equation $|M| = 0$ and $a_{1;ij}, a_{2;ij}, a_{3;ij}$ are the i-j^{th} elements of the matrices $\frac{\mathbf{C}_1^T}{(b_1-b_2)(b_1-b_3)}$, $\frac{\mathbf{C}_2^T}{(b_2-b_1)(b_2-b_3)}$, $\frac{\mathbf{C}_3^T}{(b_3-b_1)(b_3-b_2)}$, where $\mathbf{C}_s^T$ is the transposed cofactor matrix of $\mathbf{M}$ evaluated at $\beta = b_s$, for $s = 1,2,3$ respectively.

Defining the vector $\mathbf{V}(\alpha) = (V^*_{10}(\alpha), V^*_{12}(\alpha), V^*_{13}(\alpha))$, we finally have

$$\mathbf{V}(\alpha) = \mathbf{g}_1(\alpha)\cdot\mathbf{A}_1(\alpha) + \mathbf{g}_2(\alpha)\cdot\mathbf{A}_2(\alpha) + \mathbf{g}_3(\alpha)\cdot\mathbf{A}_3(\alpha)$$

where the vectors $\mathbf{g}_s = (\gamma_{10}, \gamma_{12}, \gamma_{13})/(\gamma_1 + \alpha - b_s(\alpha)r_1)$ for $s = 1,2,3$.

3.3 Queueing time of high priority class

Let the queueing time of a high priority fluid particle – representing the start of a read – be Q_1. Then the LST of the distribution function of Q_1 is

$$Q_1^*(\theta) = \mathbb{E}[e^{-\theta Q_1}] = \mathbb{E}[e^{-\theta L_1/\mu}] = L_1^*(\theta/\mu) \tag{3}$$

where L_1 is the volume of high priority fluid in the queue on arrival of the particle. This is simply the total volume of fluid present at equilibrium in the same queue when the arrival rates of fluid in all states other than 1, that of the high priority class, are zero. $L_1^*(\theta)$ is therefore given by equation 1 with fluid rate vector $\vec{r}_0 = (-\mu, \lambda_1 - \mu, -\mu, -\mu)$, as $L_1^*(\theta) = f_1^*(\theta; \vec{r}_0)/f_1^*(0; \vec{r}_0)$.

3.4 Queueing time of low priority class

Let the queueing time of a low priority fluid particle – the arrival of which represents the start of a write or erase operation in state 2 or 3 respectively – be Q_L and denote the set of valid initial states by $\mathcal{L} = \{2, 3\}$. Then Q_L is the time required to serve *all* the fluid present on arrival, of volume L_L units, say, as well as any high-priority fluid that arrives during this period – i.e. $Q_L = Z_{\ell j}(L_\ell)$ when the queueing time ends in state $j \in \mathcal{E}$, where $\ell \in \mathcal{L}$ is the state in which the low priority fluid particle arrived and L_ℓ is the fluid level at its arrival instant. Then the LST of the distribution function of Q_L is given by Theorem 2:

$$\begin{aligned}
Q_L^*(\theta) &= \sum_{j\in\mathcal{E}} \mathbb{E}[\,\mathbb{E}[\,\mathbb{E}[e^{-\theta Z_{\ell j}(L_\ell)} \mid L_\ell, \ell] \mid \ell]] \\
&= \sum_{j\in\mathcal{E}} \mathbb{E}\Big[\,\mathbb{E}\Big[\sum_{s=1}^{n_e} a_{s;\ell j} e^{b_s L_\ell} \mid \ell\Big]\Big] \quad \text{by Theorem 2} \\
&= \sum_{j\in\mathcal{E}} \sum_{s=1}^{n_e} \sum_{i\in\mathcal{L}} a_{s;ij}\, \mathbb{E}[e^{b_s L_\ell} \mid \ell = i]\mathbb{P}(\ell = i) \\
&= \frac{\sum_{j\in\mathcal{E}} \sum_{s=1}^{n_e} \sum_{i\in\mathcal{L}} a_{s;ij} f_i^*(-b_s)\lambda_i / f_i^*(0)}{\sum_{i\in\mathcal{L}} \lambda_i}
\end{aligned}$$

where $f_i^*(\theta) \equiv f_i^*(\theta; \vec{r}), i \in \mathcal{L}$ in equation 1. Although $f_2^*(\theta)$ and $f_3^*(\theta)$ are thus computed for the original model, these reflecting the initial state of the system on arrival of a low priority fluid particle, the terms b_s and $a_{s;ij}$ relate to the reduced model, with zero arrival rates for all low priority classes, since they account for the subsequent dynamics over the transient queueing period.

Considering writes and erases separately, let Q_W be the queueing time for a write operation. Then $\mathcal{L} = \{2\}$ and the preceding analysis simplifies to give

$$Q_W^*(\theta) = \sum_{j\in\mathcal{E}} \sum_{s=1}^{n_e} a_{s;2j} f_2^*(-b_s)/f_2^*(0)$$

and similarly, for erases only ($\mathcal{L} = \{3\}$),

$$Q_E^*(\theta) = \sum_{j\in\mathcal{E}} \sum_{s=1}^{n_e} a_{s;3j} f_3^*(-b_s)/f_3^*(0)$$

As an aside and a "sanity check", consider the special case where $n_e = 1, \mathcal{E} = \mathcal{L} = \{2\}$ and $\mathcal{S} = \{1, 2\}$ – not applicable to our model, of course. We find that $a_{1;22} = 1$ and $b_1(\alpha) = -\frac{\alpha+\gamma_2(1-V_{12}^*(\alpha))}{r_2}$ so that

$$Q_L^*(\theta) = f_2^*((\theta + \gamma_2(1 - V_{12}^*(\theta)))/\mu)/f_2^*(0)$$

This result could have been obtained by letting N be the number of high priority (class 1) busy periods that punctuate the time period L_L/μ. Since N is a Poisson random variable with rate parameter γ_2, abbreviating V_{12}^* by V^*, we then have

$$\begin{aligned} Q_L^*(\theta) &= \mathbb{E}[\mathbb{E}[e^{-\theta(V_1+\ldots+V_N+L_L/\mu)} \mid L_L]] \\ &= \mathbb{E}[e^{-\theta L_L/\mu}\mathbb{E}[e^{-\theta V}]^N|L_L]] \\ &= \mathbb{E}[e^{-L_L(\theta+\gamma_2(1-V^*(\theta)))/\mu}] \\ &= L_L^*((\theta+\gamma_2(1-V^*(\theta)))/\mu) \\ &= f_2^*((\theta+\gamma_2(1-V^*(\theta)))/\mu)/f_2^*(0) \end{aligned}$$

The expressions for $Q_H^*(\theta)$ and $Q_L^*(\theta)$ allow a direct comparison to be made between the latencies of the priority classes. It is straightforward to extract the moments of latency, by differentiation at $\theta = 0$, from which average performance and its variability can be compared. However, it is also possible to invert the Laplace transforms numerically, allowing information in the tails to be obtained and hence quantile-based benchmarks to be assessed.

4 Example use case and model parameterisation

For a concrete example, we consider an OLTP workload with read:write data ratio of 3:1. The random reads and random writes are both 8KB, which is the native database block size. However, we assume that the file system (such as Write Anywhere File Layout–WAFL [10]) converts all the random writes into sequential writes with negligible overhead. The Flash controller (and associated FTL) will take care of wear-levelling within Flash devices and WAFL will ensure that all Flash devices in the array are uniformly used. The OLTP application will generate traffic across the entire Flash array, but for our purposes here we consider a single Flash chip. Suppose this 32GB flash package is servicing 1600 I/O's per second (so that the Flash access density of 50 I/O's per GB is an order of magnitude better than high-speed disk drives). We also assume that the chip handles at most a single command (read, write, or erase) at a time so that power requirements for the chip are minimal. In practice, Flash controllers can pipeline several commands to different banks in the same package at the same time, though they will contend for the shared channel interface for data transfer. Given the read:write ratio of 3:1 and I/O size of 8KB, the Flash chip will see 2400 page read commands per second and 800 page write commands per second. In order to stay ahead of the write rate, the chip will also have to execute 800/64=12.5 erase commands per second (assuming 64 pages per erase block).

A concept of volume, viz. *virtual bytes* ("vbytes"), helps transform real-world problems with multiple classes and fixed service rates into fluid models. Essentially, we take one of the classes and compute the bytes transferred in an operation and hence a volume service rate based on the average service time for that class. Suppose we have a large buffer for vbytes that get serviced at a fixed rate of μ vbytes per microsecond. For Flash, a 4KB read takes 105+25 microseconds, which implies that $\mu = 31.5$ vbytes per microsecond. The assignment of vbyte volumes to the access modes is now as follows.

- to establish a reference point for the vbyte-measure of volume, when a 4KB read request is issued, we start a flow that puts 4096 vbytes into the buffer. Recalling that the buffer is drained at 31.5 vbytes/usec, this will take 130 μsecs to complete under no contention, as required;

- a 4KB write request takes 105+200μsec = 305μsecs. Although we are writing

4096 bytes of user data, we will need to put in $4(305/130) = 9610$ virtual bytes on average to simulate the actual service time of 305μsecs;

- similarly, an erase takes 1500μsecs of actual time, so we have to feed the buffer a total of $4(1500/130) = 47,262$ vbytes on average to simulate this service time.

In this way we can map between the number of virtual bytes in the request buffer and the volume in the reservoir of the fluid queue model. The constant service – or *drain* – rate is simply a transfer rate in virtual bytes per microsecond. During periods of no contention, a command starts an operation mode (read, write or erase) with flow rate that typically will be greater than the constant drain rate (31.5 vbytes/μsec).[3] The completion of that mode's active period determines the volume added to the reservoir, proportional to the specified duration. Notice that the instantaneous rate at which vbytes enter the buffer is arbitrary to some degree: a mode's long term input rate is preserved if the product of its average duration in unit time and its instantaneous flow rate remains constant. Higher rates reflect increased burstiness, as discussed below.

For the modulating CTMC, we also have to estimate the mean state holding times, along with the transition probabilities between states. These parameters are necessary to construct the generator matrix and can only be obtained by monitoring workloads typical (in this use case) for Flash devices with OLTP workloads; see [18] for example. In particular, greater burstiness can be represented by decreasing state holding times (increasing their out-transition rates) and correlation in sequences of command-mode types can be accounted for by the state transition probabilities.

5 Numerical results

5.1 Fluid model parameterisation and queueing times

The model detailed in section 3 was applied to the use case of the previous section as a preliminary illustration of its capability. The rates of fluid input in each command mode and drain rate, measured in vbytes per microsecond, were set as described in the previous section. The CTMC's transition probability matrix was set, somewhat arbitrarily in the absence of a reliable OLTP workload model, to:

$$\begin{pmatrix} 0 & 0.74 & 0.25 & 0.01 \\ 0.74 & 0 & 0.25 & 0.01 \\ 0.25 & 0.74 & 0 & 0.01 \\ 0.1 & 0.148 & 0.752 & 0 \end{pmatrix}$$

The instantaneous transition rates out of each state were set to $(2, 5, 2, 10)/1000$. These choices reflect plausible correlations and reasonable burstiness in a typical OLTP sequence of database accesses. Moreover, it yields correctly the long term average numbers of vbytes issued by each command type in one microsecond, *viz.* 9.83, 7.69, 0.59 for reads, writes and erases, respectively. These figures become the required vector $(2400, 800, 12.5) \times 10^{-6}$ when scaled down by the vbyte-factor 4096 and by the single command relative access time vector $(1, 305/130, 1500/130)$. The instantaneous vbyte-arrival rates are now obtained by dividing the long term average vbyte arrival rate vector by the equilibrium state probability vector of the CTMC with generators specified above, which turns out to be $(0.4929, 0.2284, 0.276, 0.0027)$. This gives instantaneous vbyte-arrival rates of $0, 43.0, 27.9, 220.5$.

[3] At least one mode's flow rate must be greater than the drain rate, otherwise the reservoir would always be empty. However, some modes *may* have a flow rate lower than the drain rate, as in our reduced model, for example, where it is zero for erases and writes.

In this scenario, the flash unit is running at a moderate utilisation of 57.5% (average total arrival rate, 18.1, divided by the drain rate, 31.5) and so we can expect relatively small fluid levels and access times at equilibrium. In fact, differentiating equation 1, we find that the mean fluid level is 2364.4 (about half a read access) and its standard deviation is 7202.6 – suggesting fairly long tails. For the high priority reads, from equation 3, we obtain the queueing time probability density function shown in figure 2 (left), with mean queueing time 82.19μsecs. For the low priority writes/erases, we get the density function shown in figure 2 (right), with mean queueing time 902.2μsecs. This is broadly what we would expect at utilisations of around 50%.

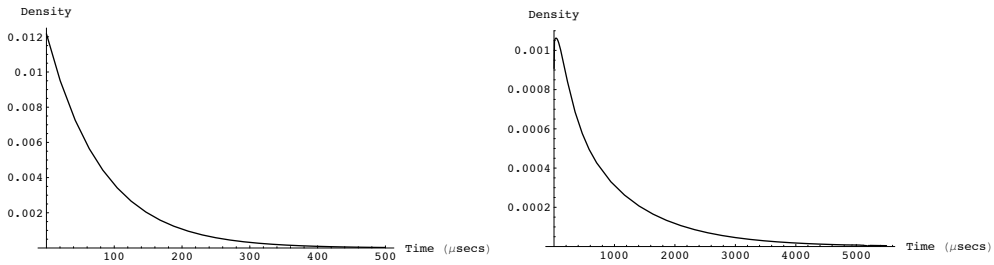

Figure 2: Queueing time density at moderate utilisation for high priority reads and low priority writes/erases: mean values 82.19 and 902.2 μsecs respectively

The two densities are also shown together on the same axes in figure 3 to illustrate the difference in scale at high and low utilisations.

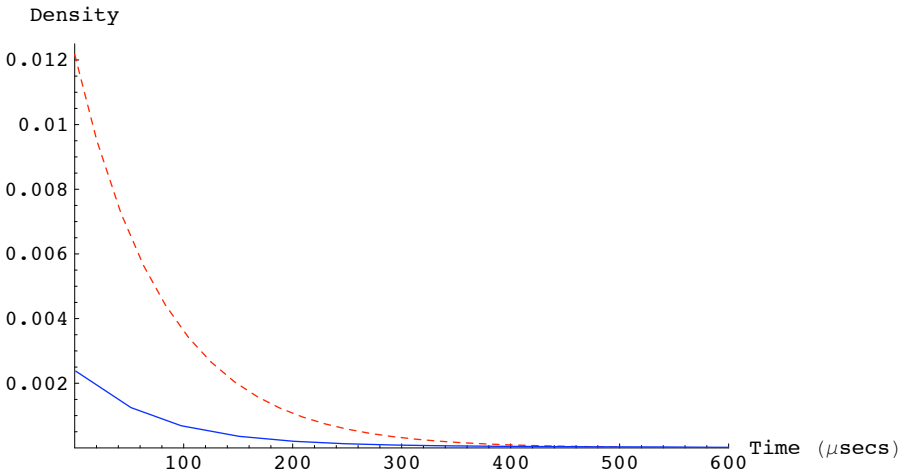

Figure 3: Queueing time densities for high (dashed, red) and low (solid, blue) priority requests: moderate utilisation

To examine the effect of increased workload, we scaled all the arrival rates up by a factor 1.6, giving a new total average arrival rate of 28.97 vbytes/μsec and utilisation of 92%. The mean fluid level is now 85328.3 and its standard deviation is 96225.3. Mean queueing times grow to 365.6 and 5907.0 μsecs for the high and low priority classes respectively, and the densities are shown in figure 4. The difference

in scale is now magnified by a factor of about five so that the high priority density function is barely distinguishable from part of the vertical axis. The dual graph is therefore not shown.

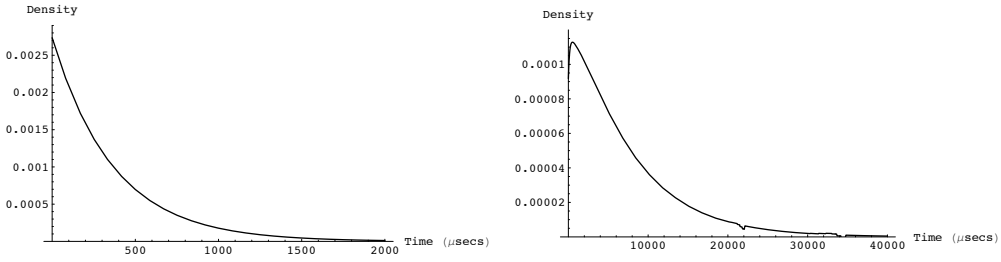

Figure 4: Queueing time density at high utilisation for high priority reads and low priority writes/erases: mean values 365.6 and 5907.0 μsecs respectively

5.2 Model validation by simulation

We wrote a customised event driven simulator in C to represent the Flash storage system and its associated modes of operation. The simulator handles a specific architecture for the current study but it can easily be extended to alternate Flash configurations by using their description files available in hardware libraries. In the present case, the target architecture is a package of 16 K9KAG08U0M NAND-Flash of 2GB [4] each, connected by a single 40MBps channel. This Flash storage package is seen as a single address space, logical storage pool. Data is manipulated using three commands – read, write and erase – one at a time. The service times of these commands were estimated as constants by measurement on real chips. The simulator's execution module processes events as they occur – for example the arrival of a request of a given type or the completion of an access – by maintaining a standard event diary. Both preemptive and non-preemptive priority modes can be accommodated, but here we consider only the preemptive variant, to be consistent with the fluid model. The simulator ensures correct implementation of the relative priorities between the two classes: a high priority class composed of reads only and a low priority class composed of writes and erases. For the preemptive mode, a read is never interrupted whereas a write or an erase (low priority event) is launched only if no reads are in the system and is interrupted as soon as a read enters the system. The interrupted write or erase is resumed as soon as there are no reads in the system again, but subject to further interruption.

Simulations were run using different workloads, consistent with the stated OLTP criteria, e.g. the fixed read:write ratio of 3:1. Various request arrival rates were used, yielding package utilisation percentages ranging from 40% to 92%. The performance metric considered was again the mean queueing time of a request and a set of five MMPP traces, specifically generated as input to the simuator, was used to give five estimates at each arrival rate considered. This was to facilitate a batched means estimation of confidence bands [11]. Each trace had 100,000 request arrivals, the first 10,000 being discarded, which ensured the system had reached (close to) equilibrium. The results obtained are presented and discussed in the next subsection.

[4] http://www.samsung.com/global/system/business/semiconductor/product/

5.2.1 Model Validation

A constant operation service time is one of the defining characteristics of Flash and so queueing time and its variability as the package utilisation changes are quantities that have a significant impact on overall Flash performance. We focus our attention here on the mean queueing time for each of the two classes (reads and writes-erases) at different arrival rates, and so at different utilisations. MMPP-generated workload traces corresponding to specific sets of the model's parameters were used as input to the simulator. The results obtained by each model (fluid and simulation) therefore relate to the same context. Figures 5, 6, 7 and 8 show the queueing times obtained by simulation (dashed, red plots) compared with the fluid model's predictions (blue, unbroken plots) for requests of high priority (reads) and low priority (writes and erases), respectively. We also show 95% confidence bands on the simulation plots. We see reasonable agreement in the first two graphs, which relate to the model parameterisation used in section 5.1. For the reads, the graphs have exactly the same shape for both the simulation and the model, but they are separated by a small vertical shift. This is likely because the reads do not constitute "heavy traffic", which is the domain in which fluid models tend to perform most accurately. In fact, the difference between these curves is less than one read service time (130 microseconds) and so could be accounted for largely by the difference in the models' granularities.

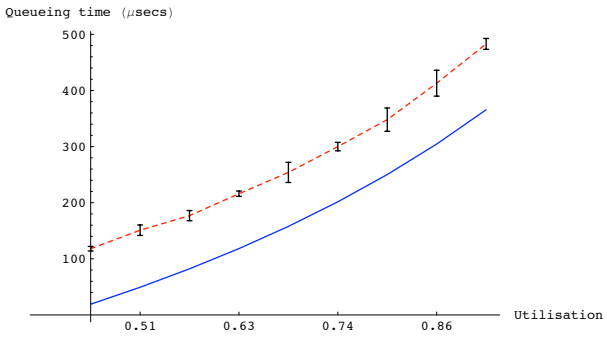

Figure 5: Queueing time for high priority class

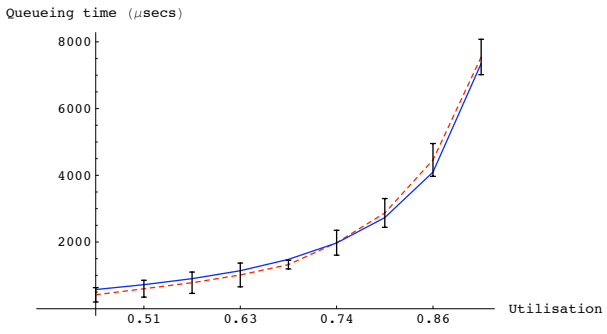

Figure 6: Queueing time for low priority class

We therefore increased the burstiness of the input traffic by modifying the state transition rate vector of the CTMC to $(1, 7, 3.5, 3)/1000$. This increased the arrival

rates in each state of the CTMC considerably, it's off-state now having equilibrium probability $\pi_0 = 0.75$. The resulting graphs showed much better agreement, although the simulation showed signs of not having converged adequately in the case of the most heavily loaded low priority class (i.e. at the rightmost point in Figure 8).

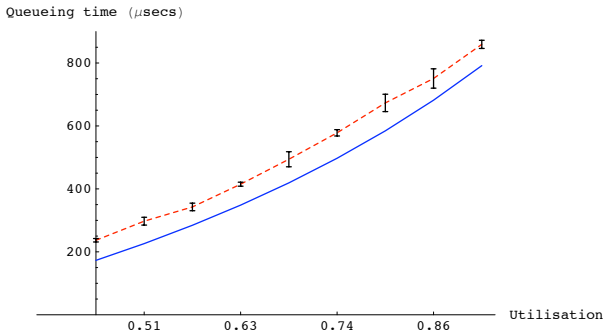

Figure 7: Queueing time for high priority class at higher burstiness

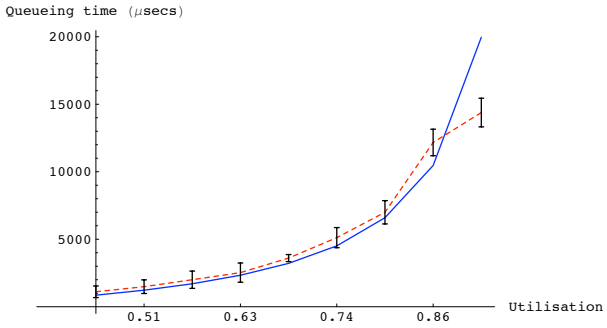

Figure 8: Queueing time for low priority class at higher burstiness

6 Conclusion and future work

With some simplifiying assumptions, we have shown that fluid modelling techniques provide a promising tool for the performance evaluation of NAND Flash memory systems. This approach has a considerable advantage over traditional queueing models in that it can account for both correlated input and priorities. Moreover, under heavy load, it is well known that fluid models tend to be accurate, often a stochastic limit of an analogous queue. The preliminary numerical results are encouraging, showing the right qualitative behaviour, and testing against simulation provided good quantitative validation. Notice that each point in the graphs of the mean values in the previous section took over an hour to obtain by simulation; the corresponding complete graph required only seconds to construct from the fluid model. Further assessment against experimental observations in a wider range of scenarios is, of course, crucial in the short term.

We intend to extend the fluid model developed here, which has a preemptive priority policy, to handle non-preemptive policies that better match current Flash

devices. Indeed a comparison of FCFS, preemptive resume, and non-preemptive policies would show the impact of the long erase times. As Flash devices become denser, erase times are expected to increase and even if erases are done in the background, it is possible that a long sequence of writes will use up all available free space and force erases that could delay writes. This motivates the extension of our model to investgate these effects. On the hardware side, currently Flash memory represents the lowest level in the hierarchy. The shared channel interface and flash dye are modelled as one resource, but in fact they are separate devices that are held simultaneously during command and data transfers. Another interesting extension would be to handle command queueing inside the Flash device so that more than one memory bank could be active at any one time. Then, we can exploit the parallel access to Flash banks in order to improve the overall Flash system performance. We also intend to consider a set of Flash devices, just like disk drives, and investigate data striping and related parallel access policies, using analytical modelling and validation with respect to simulation.

Finally, we will consider alternative workloads, representing other environments taken from the wide range that use Flash devices. This will facilitate the prediction of Flash storage system behaviour under the demands of a variety of applications. A parsimonious way to describe such workloads is to use hidden Markov models (HMMs). From these a mapping onto a Markov modulated fluid arrival process can be defined, providing the generators of the CTMC in a systematic way, thus avoiding the arbitrariness noted in section 5. Preliminary results on this complete modelling pipeline may be found in [19].

References

[1] Leo Freitas Andrew Butterfield and Jim Woodcock. Mechanising a formal model of flash memory. *Science of Computer Programming*, 74, 2009.

[2] Andrew Butterfield and Jim Woodcock. Formalising flash memory: First steps. In *12th IEEE International Conference on Engineering Complex Computer systems*, pages 251–260, 2007.

[3] Li-Pin Chang and Tei-Wei Kuo. An adaptive striping architecture for flash memory storage systems of embedded systems. In *Proc. Eighteen IEEE Real Time and Embedded Technology and Application Symposium (RTAS'02)*, 2002.

[4] Li-Pin Chang and Tei-Wei Kuo. Efficient management for large scale memory storage systems. In *Proc. ACM Symposium On Applied Computing*, 2004.

[5] D. Mitra D. Anick and M. Sondhi. Stochastic theory of a data-handling system with multiple sources. *Bell Syst. Tech. Journal*, 8(61), 1982.

[6] Hynix Semiconductor et al. Open nand flash interface specification. Technical report, ONFI, December 2006. www.onfi.org.

[7] Anthony G. Field and Peter G. Harrison. An approximate compositional approach to the analysis of fluid queue network. *Performance Evaluation*, 64, 2007. Proceedings of Performance 07.

[8] Marco Gribaudo and Miklos Telek. *Fluid Models in Performance*, volume 4486. SpringerLink, 2007.

[9] Peter G. Harrison. A tandem network of fluid queues with on-off arrivals. Technical report, Department of Computing, Imperial College London, 2008.

[10] Dave Hitz, James Lau, and Michael Malcolm. File system design for an nfs file server appliance. In *WTEC'94: Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, pages 19–19, Berkeley, CA, USA, 1994. USENIX Association.

[11] Raj Jain. *The Art of Computer Systems Performance Analysis:Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley Interscience, 1991.

[12] Dirk Botle Jorn Engel and Robert Mertens. Garbage collection in logfs. In *Linyx conf.*, 2007.

[13] Hyojun Kim and Seongjun Ahn. Bplru: A buffer management scheme for improving random writes in flash storage. In *Proc. 6th USENIX Conference on File and Storage Technologies*, 2008.

[14] Seung-Ho Lim and Kyu-Ho Park. An efficient nand flash file system for flash memory storage. *IEEE Transactions On Computers*, 55(7), 2006.

[15] Aleph One Limited. Yaffs2 readme-linux v1.1, 2007. http://aleph1.co.uk.

[16] LogFS. Logfs home page. http://logfs.org/logfs/LogFS.

[17] Charles Manning. Introducing yaffs, the first nand-specific flash file system, 2002. http://www.linuxdevices.com/articles.

[18] Naresh M. Patel Peter G. Harrison and Soraya Zertal. Response time distribution of flash memory accesses. In *Proc. ValueTools*, 2008.

[19] N.M. Patel P.G. Harrison, S.K. Leyton and S. Zertal. A hidden markov model for flash storage workloads. Technical report, Imperial College London, May 2009.

[20] David Woodhouse. Journaling flash file system (jffs) and journaling flash filesystem 2 (jffs2). http://sources.redhat.com/jffs2/jffs2-html.

[21] Chin-Hsien Wu and Tei-Wei Kuo. The design of efficient initialization and crash recovery for log-based file systems over flash memory. *ACM Transactions on Storage*, 2(4), 2006.

[22] Chul Lee Youngwoo Park, Seung-Ho Lim and Kyu Ho Park. Pffs: A scalable flash file system for the hybrid architecture of phase-change ram and nand flash. In *2008 ACM Symposium onApplied Computing*, pages 1498–1503, 2008.

[23] Jen-Wei Hsieh Yuan-Hao Chang and Tei-Wei Kuo. Endurance enhancement of flash memory storage systems: An efficient static wear levelling design. In *Proc. DAC*, June 2007.

[24] Soraya Zertal. A reliability enhancing mechanism for a large flash embedded sattelite storage system. In *Proc. IEEE International Conference on Systems (ICONS)*, 2008.