

Dynamic File Migration to Support Parallel Database Systems

J.A. McCann, N.J. Williams, P. Osmon, D. Bolton

Systems Architecture Research Centre
CITY University, LONDON, EC1V 0HB, ENGLAND

Abstract

Research into DBMS (Database Management System) parallelism has been carried out to address the performance problems experienced in areas such as Decision Support. Distributed shared memory can alleviate the porting of commercial DBMSs to parallel platforms. However, this restricts the scalability and performance achievable using shared nothing approaches. Consequently we propose the use of dynamic data migration techniques to support such parallel DBMSs. Using simulation techniques, we examine the effects which parameters such as migration threshold, migration block size and degree of data sharing have on file system performance. We present simulation results, providing useful insight into file system design.

Project Background

The database user community is continuously demanding highly usable and portable products which exhibit good performance. Consequently, extensive research into the use of parallelism has been carried out to resolve these performance problems. Moreover, leading researchers assert that highly parallel database systems, based on standard general purpose architectures, are a prime solution to alleviate inadequate DBMS performance [1, 2, 3, 6].

Porting an established DBMS product to a parallel machine is a formidable task due to the expanse of code involved, and the fact that such code has evolved over time. It has been reported that many person years are involved in the laborious rewriting of serial code to exploit parallelism [8]. It is therefore our objective to not only port a commercial DBMS product to a parallel machine, but to do so in a generic way, making minimum changes to the original code. In this project we aim to increase the performance of DBMS processing through parallelism, whilst demonstrating that the complexities of porting a commercial DBMS are reduced for a distributed machine consisting mostly of standard components supporting distributed shared memory. Our current research introduces the Ingres database system to Topsy; a

message-passing multicomputer, comprising a distributed UNIX-compatible kernel called Meshix (System V release 3.0 with BSD extensions) [5].

It is less costly to communicate queries as opposed to data [1]. Therefore due to the constraints of our architecture Distributed Virtual Shared Memory (DVSM), we have decided to adopt what could be deemed a hybrid paradigm in our design. That is, from the database system point of view we are emulating a shared nothing system, where data is partitioned over the disks, and the query sent to the closest processor for execution. On the other hand, from the lower levels and filesystem viewpoint we essentially have shared memory and disks. Throughout the project's lifetime performance modelling techniques have been successfully employed to speed up the port and aid in the design decision making process.

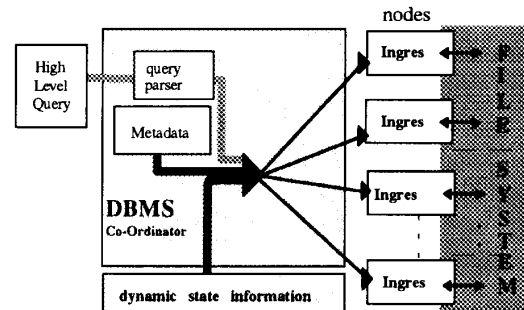


Figure 1. Parallel DBMS design with co-ordinator and dynamically migrating file system

Parallelising A Commercial DBMS

To abate the performance degradation that would be experienced in a distributed shared memory environment, a data migration policy at the file system level has been proposed. Essentially the DBMS (Ingres) is sandwiched between a Co-ordinator module and a file service performing dynamic data migration. This is shown graphically in figure 1. The Co-ordinator handles the DBMS query parallelisation; scheduling and query optimisation, based upon logical domain partitioning.

Traditional scheduling and data partitioning techniques are implemented[9,10]. To obtain the required performance benefits, the filesystem must therefore map the logical partitioning of data to the physical data partitioning. Hence the use of dynamic data migration techniques which, after a 'warm-up' period, will ensure data is resident on the node making the greatest demand for that data.

Why Migration?

Providing a high degree of locality of data in a distributed system is obviously beneficial. Several methods exist that have been used extensively in file system development, although migration is generally neglected. The aim of achieving perfect locality is most commonly provided through the use of cache technology. The obvious drawback of this system is the unpredictability of the caches' contents, particularly as the file service will be sharing the cache with other applications' data. Increasing the cache size helps this problem but does not cure it; it is a non-scalable solution. To improve on the behaviour, techniques such as pre-fetching can be invaluable: reducing the data latency to the minimum. However with a multitasking operating system, relying on the contents of the cache can be erroneous, no matter how large the cache. Our study differs from previous studies in that we examine the movement of partial files, varying the amount.

When the cache is ineffective, it becomes necessary to retrieve the data from the source, often significantly increasing the latency. This data latency is frequently a poorly addressed problem, typically solved by providing data replication within the file service or by dismissing the latency as irrelevant. However, we contend that the overhead in maintaining replicated data is higher than the alternative of data migration, especially with the large volumes of data such as that used within decision support applications. The purpose of data migration is to move the data around the distributed system bringing it as close as possible to the requesting application. Then when the cache does not hold the required data, it will be found on disks local to the application. As the migration of the data itself is demand-based, the cost of the migration is negligible: the data must be transferred to the local cache anyway. The only overhead incurred is in the transferral of ownership of the data: a relatively small penalty of updating meta-information as to the source of the data. As applications will have different behaviour when accessing the data in the file system, it becomes necessary to allow the application to define any parameters governing the migration mechanism, even to the extent of disabling migration completely; we assume that the application has

a better idea of its own data access patterns than the file service will.

Design Evaluation and Model Assumptions

There are a number of issues involved in evaluating the effectiveness of a migrating file system. Consequently we have developed a model which simulates data access varying parameters such as migration threshold and block size, measuring their affects on performance. Fundamentally, we wish to address the question: "how many remote requests to a data block constitutes it to be a candidate for migration and, what is the ideal migration block size at that point?"

A simulation model, parameterised with empirical measurements, has been used to evaluate the proposed migrating file system. Simulation techniques are ideal when modelling file systems because areas such as queueing, contention and hardware utilisation can be readily measured, which is lacking in some data migration studies[7]. A simulation package, Network II.5, has been applied to assist in model development.

We have modelled, and measured a four node Topsy file system. At the hardware level, files are stored on physical disks, in strings of data blocks. Table 1 gives an example of the model's parameters, which are varied corresponding to the characteristics of general application behaviour as well as typical DBMS behaviour. Two important parameters are the average local data transfer unit time and the remote transfer unit time. Both these parameters were obtained tracing access times during a typical day's operation on the Topsy machine, and the resulting distribution is then passed into the simulation.

To reiterate, the aim is to match the process migration in our parallel DBMS to the data migration of the file system, thus achieving shared nothing performance. The probability of data being local in dynamically migrating file systems is dependent on three major factors: the degree of data sharing, migration threshold and migration block size.

Locality Behaviour.

The degree of data sharing (d) is the probability that more than one node will request the same data block. This reflects the parallelism within the application and therefore is the most significant of all the parameters. In our model we represent this in terms of how d affects the probability of data being local $p(l)$. That is, when $d=0\%$, once a block has been migrated to a particular node, no other nodes will subsequently request it. Therefore the probability of a data block being local will depend on how much of the file partition has been migrated to this particular node and this increases as more of the data migrates. Whilst we confidently assume that the degree of

data sharing exhibited by the DBMS applications in a shared nothing environment will not be greater than 5%, for completeness we model a wider range of degrees of data sharing.

Migration Threshold and Migration Block Size

Migration threshold, m is the number of remote requests for a particular data block which should be received before the data block will be migrated to the requesting node. If we have a high degree of data sharing and a low migration threshold, we may experience the thrashing typically found in previous migrating file systems[7]. Again we have to trade off remote requests for data blocks against the delay caused when migrating a data block.

Another major factor to be taken into account is the migration block size, b . This is the size of the data block migrated between nodes. This is not necessarily the same size as the standard data access block and may vary up to the complete size of the file. In our model we vary the migration block size in terms of its percentage of the complete partition size. The larger the block size the more local the data will become as it is migrated, but this is offset by the fact that a higher cost is incurred when migrating a larger block. Moreover, with high data sharing and a high migration block size we may again observe a high degree of thrashing of a very large file partition which will cause much contention and further delays.

Taking these factors into account we derive the following equation to give us the probability of executing a local request: (percentage of file migrated is determined by the migration block size and number of migration blocks already migrated).

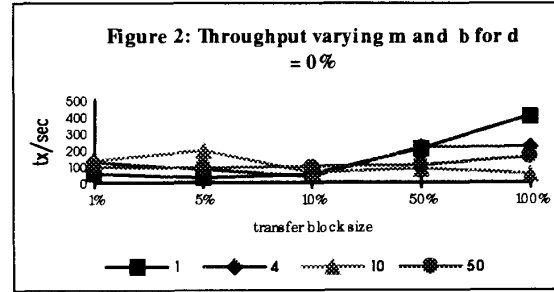
$$p(l) = \% \text{ partition migrated} * (1 - p(d))$$

The probability that the next block will always be local ($p(d)=0\%$, $p(l) = 100\%$), occurs when 100% of the file partition has migrated to the node.

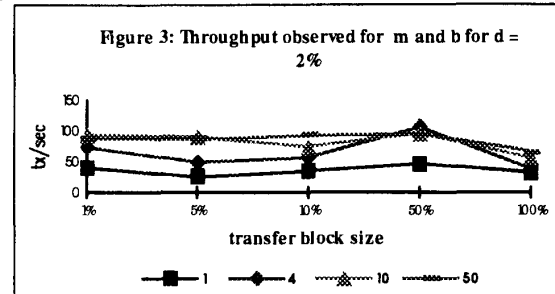
Experiment Results

Here we present these results varying the migration threshold (m) and transfer block size (b), for a number of degrees of data sharing (d).

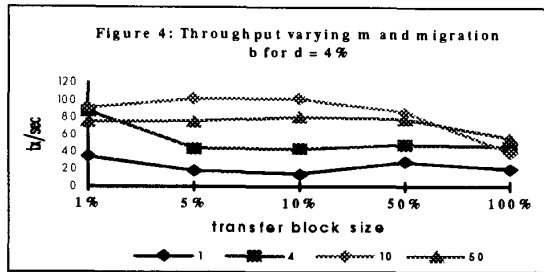
The degree of data sharing is varied respectively for each graph. The extent to which data is shared between nodes is very much application dependent, though the expected maximum degree of data sharing will be less than 5% using our Co-ordinator with the DBMS. For each degree of data sharing we vary the migration threshold, while varying the migration block size. The results from these simulations are presented in figures 2 through to 6.



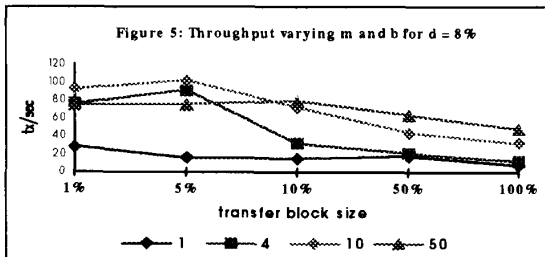
From figure 2 we observe that when there is no data sharing, the parameters providing the best throughput is that of $m=1$ and $b=100\%$. That is, after a single remote request we migrate the complete file partition to the requesting node so that further requests may access the data locally. Also, if we increase the migration threshold to as much as $m=10$ we observe a great decrease in throughput. This consequently demonstrates that holding off migration in the shared nothing case causes a large number of remote reads which degrade performance to an extent greater than having to migrate the file partition. On the other hand, it is also interesting to note the behaviour for a 20 second simulation time at the points where $m < 5$ and $b > 50\%$. In these cases we observe very poor load balance and find that two nodes provide (with processor utilisation $< 10\%$) and the remaining two nodes process the data (processor utilisation $> 93\%$).



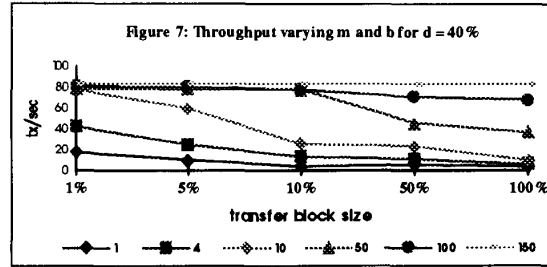
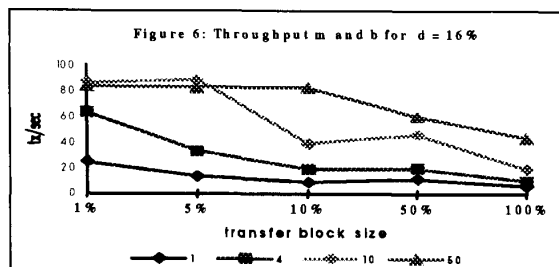
With a 2% chance of data being requested by another node, we observe slightly different results shown in figure 3. Firstly we can observe the performance degradation resulting from the decrease in parallelism (that is, the increase in data sharing). Consequently, moving the complete file partition ($b=100\%$) gives very low throughput no matter what the migration threshold is. This is due to the high overhead incurred by moving the complete file partition between nodes a number of times. The graph shows that the best throughput is gained by moving half of the file partition ($b=50\%$) but deferring the migration, after an increased number of remote requests, i.e. 4. Figure 3, highlights the question as to whether migration is indeed necessary at all when we have a chance of data being shared between nodes. The next set of experiments investigate this further.



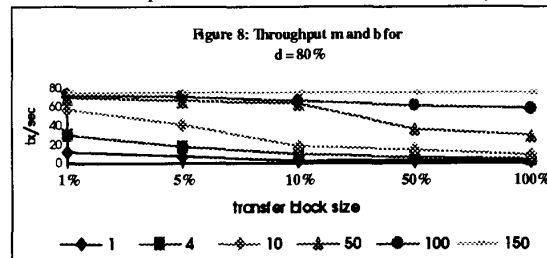
Data sharing is then doubled to 4%, and the results shown figure 4. In this case the parameters providing the best throughput are a migration block size of 5% after 10 requests. In fact, we can observe that most migration block sizes produce better performance with a migration threshold of 10. When the degree of data sharing doubles to 8% (figure 5), complete file partition migration now exhibits the lowest performance. As before, better throughput is experienced again transferring data units of 5% of the total data partition after 10 requests. This follows for 16% data sharing, where we notice that again further doubling the probability of data sharing has not affected the other parameters producing the best overall throughput.



Whilst a degree of data sharing of 16% is way beyond that expected from our database system, it is interesting to note that there seems to be a trend related to deferring migration and the increasing the degree of data sharing. To this end, two further sets of experiments were carried out, increasing d to a large number. The results are presented in figures 7 and 8 respectively. The resulting maximum throughput observed when an application has an 80% degree of data sharing is 74.9 transactions per second. When examining the results further we discover that in fact no migration was carried out during our



simulation period; this is effectively performing remote reads. This assumption was further validated running the same simulation model with no migration strategy. Therefore we can conclude that parameters which produce a throughput of less than 74.9 transactions per second are in fact thrashing the system to an extent that it would be more productive to read the data remotely.



Summary

To summarise, we can clearly see the affects data sharing has on the maximum throughput obtainable. We charted the throughput observed in a non-migrating system against the best of the migrating file system's throughput with equal degrees of data sharing. Immediately it can be seen that in the worst case (80% data sharing) the performance is equal, and in the best case (no data sharing) we can achieve up to four times the throughput, see below.

Database Parallelism will demonstrate its greatest benefit in Decision Support type applications. These typically involve complicated queries, acting on large volumes of data, with no updates. Consequently, our model's results could be deemed a pessimistic view of what is expected from general file system operation. That is, traditionally dynamic file migration demonstrates its

d	m	b	optimum tx/sec	tx/sec no data migration
0	1	100	410	101.5
2	4	50	106.3	87.35
4	10	5-10	102.2	86.05
8	10	4	101.65	81.55
16	10	5	89.2	75.1
40	150	doesn't matter	83.65	75
80	150	doesn't matter	74.9	75

true usefulness in applications which contain a large number of writes, which have not been modelled in this exercise.

Conclusions

The use of simulation modelling has clearly supported the quick design and development of a project such as that presented in this paper. The findings resulting from this model aid the file system designer, but more importantly a more effectual use of the results will be in the heuristics which govern the on-line migration system. An estimate of the degree of data sharing exhibited by an application can be combined with a number of other application parameters such as communications latency, and used to control data placement and migration.

Simulation techniques have proven to be useful in terms of quickly building complicated models combined with low simulation times, which have encouraged the excessive experimentation required to reach our conclusions. We have shown that the non-deterministic nature of file system behaviour such as contention and queueing can be modelled producing a wealth of results. That is, we were able to carry out more than 100 experiments, varying parameters, each taking only a few minutes to run. More importantly we have been able to demonstrate the extent to which dynamically migrating file systems can help increase performance of distributed and parallel applications. Especially suited are parallel database systems, for shared nothing architectures which exhibit degrees of data sharing much less than 16%. We examine the major parameters which affect file system performance, using simulation modelling to evaluate their effects. Consequently we provide valuable insight into migrating file system design, while also defining parameters which can characterise applications for the file system's usage.

References

1. DeWitt D, Gray J., 'Parallel Database Systems: The Future of High Performance Database System', Communications of the ACM, Vol. 35, No.6, 1992
2. Hanson J G, Orooji A, 'Predictive Performance Analysis of a Multi-Computer Database System', Information Systems, Vol. 15, Part. 4, 1990
3. Kiessling K, 'Access Path Selection in Databases With Intelligent Disk Subsystem', The Computer Journal, Vol. 31, No.1, 1988
4. McCann J.A., ' Report on Distributed File System Migration and Caching'. Technical Reports, August - November 1993
5. Osmon P, Stiemerling T, Valsamidis A, Whitcroft A, Wilkinson T, Williams N, 'The Topsy Project: a position paper'. In PARLE 92. Paris: Springer-Verlag.,1992
6. In Parallelogram, p 4, August 1991
7. Stumm M., Zhou S., 'Algorithms Implementing Distributed Shared Memory'. Computer, May 1990
8. Fox A., Personal communication with Aubrey Fox, 27 April 1994, ASK Ingres
9. Ghandeharizadeh et al, ACM-SIGMOD 1992
10. Srivastava et al IEEE-Data Engineering, 1993

Hardware	Software
processor speed	probability and size of local processor requests
transfer device speed, connections	number and size of local ack messages
block overheads	probability and size of remote processor requests
routing links	number and size of remote ack messages
storage devices, disk capacity	probability of local cache hits/misses
cache capacity	probability of page swap/dirty page written
	File size in terms of data transfer units
	remote data transfer unit times
	local data transfer unit times
	migration threshold
	migration block size

Table 1. Some variable input parameters for the filesystem model.