How to determine a good multi-programming level for external scheduling

Bianca Schroeder§ Mor Harchol-Balter§* Arun Iyengar† Erich Nahum† Adam Wierman§
§Carnegie Mellon University
Department of Computer Science
Pittsburgh, PA USA
<bianca, harchol, acw>@cs.cmu.edu

†IBM T.J. Watson Research Center
Yorktown Heights, NY USA
<aruni,nahum>@us.ibm.com

Abstract

Scheduling/prioritization of DBMS transactions is important for many applications that rely on database backends. A convenient way to achieve scheduling is to limit the number of transactions within the database, maintaining most of the transactions in an external queue, which can be ordered as desired by the application. While external scheduling has many advantages in that it doesn't require changes to internal resources, it is also difficult to get right in that its performance depends critically on the particular multiprogramming limit used (the MPL), i.e. the number of transactions allowed into the database. If the MPL is too low, throughput will suffer, since not all DBMS resources will be utilized. On the other hand, if the MPL is too high, there is insufficient control on scheduling. The question of how to adjust the MPL to achieve both goals simultaneously is an open problem, not just for databases but in system design in general. Herein we study this problem in the context of transactional workloads, both via extensive experimentation and queueing theoretic analysis.

We find that the two most critical factors in adjusting the MPL are the number of resources that the workload utilizes and the variability of the transactions' service demands. We develop a feedback based controller, augmented by queueing theoretic models for automatically adjusting the MPL. Finally, we apply our methods to the specific problem of external prioritization of transactions. We find that external prioritization can be nearly as effective as internal prioritization, without any negative consequences, when the MPL is set appropriately.

1. Introduction

Many of todays web applications are largely dependent on a backend database, where the majority of the request processing time is spent. For such applications it is often desirable to control the order in which transactions are executed at the DBMS. An e-commerce applications for example might want to give faster service to those transactions carrying a lot of revenue.

Recently, systems researchers have started to investigate the idea of *external scheduling* as a method of controlling the order in which transactions are executed. The basic mechanism in external scheduling is demonstrated in Figure 1, and simply involves limiting the number of transactions concurrently executing within the DBMS. This limit is referred to as the MPL (multi-programming limit). If the MPL is already met, all remaining transactions are queued up in an external queue. The application can then control the order in which transactions are executed by scheduling the external queue.

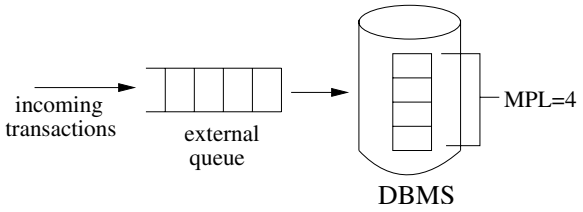


Figure 1. *Simplified view of the mechanism used in external scheduling. A fixed limited number of transactions (MPL=4) are allowed into the DBMS simultaneously. The remaining transactions are held back in an external queue. Response time is the time from when a transaction arrives until it completes, including time spent queueing externally to the DBMS.*

Examples of recent work on external scheduling come from many areas including storage servers, web servers, and database servers. For example, Jin et al. [9] develop an external scheduling front-end to provide proportional sharing among the requests at a storage service utility. Blanquer et al. [4] study external scheduling for quality of service pro-

*Supported by NSF grants CCR-0133077, CCR-0311383, 0313148, and a 2005 Pittsburgh Digital Greenhouse Grant.

visioning at an Internet services cluster. In our own recent work [22] we propose external scheduling for providing class-based quality of service guarantees for transactional, database driven workloads. Finally, for many commercial DBMS there exist tools that provide mechanisms for external scheduling, such as the IBM DB2 Query Patroller [2].

The advantage of the external approach is that it is portable and easy to implement since it does not require changes to complex DBMS internals. Moreover it is effective across different types of workloads, since (unlike the internal approach which directly schedules the resources inside the backend DBMS) external scheduling works independently of the system's bottleneck resource. It is also very flexible in that it allows applications to implement their own custom-tailored scheduling policy, rather than being limited to the policies supported by the backend DBMS.

While the basic idea behind external scheduling is simple, its efficacy in practice hinges on the right choice of the MPL. For scheduling to be most effective a *low* MPL is desirable, since then at any time only a small number of transactions will be executing inside the DBMS, while a large number are queued under the control of the external scheduler. On the other hand, *too low* an MPL can hurt the overall performance of the DBMS, e.g., by underutilizing the DBMS resources resulting in a drop in system throughput. While many have cited the problem of choosing the MPL in external scheduling as critical, previous research in all areas of system design leaves it as an open problem. Existing tools for external scheduling leave the choice of MPL to the system administrator.

The question of this paper is: How low can we choose the MPL to facilitate effective scheduling, without hurting overall system performance? There are three important considerations when choosing an MPL: (1) As already mentioned, by holding back transactions outside the DBMS, the concurrency inside the DBMS is lowered, which can lead to a drop in throughput. We seek algorithms that determine, for any input scenario, the *lowest* possible MPL value necessary to ensure near-optimal throughput levels (when compared to the system without MPL). (2) Holding back transactions, and sequencing them (rather than letting them all share the database resources concurrently), creates the potential for head-of-line (HOL) blocking where some long-running transactions prevent other shorter transactions from entering the DBMS and receiving service. This can result in an actual increase in overall mean response time. We seek algorithms that determine, for any input scenario, the *lowest* possible MPL value necessary to prevent an increase in overall mean response time. (3) Lastly, it is not at all obvious that external scheduling, even with a sufficiently low MPL, will be as effective as internal scheduling, since an external scheduler does not have any control over the transactions once they're dispatched to the DBMS.

Section 2 describes the wide range of hardware configurations, workloads and different DBMS we use in our experiments. Section 3 evaluates experimentally how low we can set the MPL without hurting throughput and overall mean response time. We find that the answer to this question is complex, and we identify the dominant factors that provide the answer to this question. Next, in Section 4 we create queueing theoretic models based on the findings in Section 3, that capture the relationship between the MPL and throughput and overall mean response time. We then show how a feedback-based controller can be used, in conjunction with the queueing models, to automatically adapt the MPL. Finally, in Section 5 we evaluate the effectiveness of external scheduling in one particular application involving prioritization of transactions. We study whether external scheduling with the appropriately chosen MPL can be as effective as internal scheduling with respect to providing differentiation between high and low priority transactions.

It is important to note that throughout this paper the question is how *low* an MPL one can choose without hurting system performance. While this question has not been addressed in any previous work, a complementary question involving *high* MPLs has been looked at in the context of admission control, see for example [5, 8, 10, 12, 18]. The point of these studies is that throughput suffers when too many transactions are allowed into the DBMS at once, due to excessive lock contention (lock thrashing) or due to overload of some system resource. Hence it is beneficial to have some high MPL upper bound on the number of transactions allowed within the DBMS, with the understanding that if this MPL is set too high, then throughput will start to drop. Admission control studies how to limit the number of concurrent transactions within the DBMS by *dropping* transactions when this limit is reached. Our work looks at the other end of this problem – that of very low MPLs needed to provide prioritization differentiation or some other type of scheduling – and does not involve dropping requests.

2. Experimental setup

To answer the questions of feasibility and effectiveness of external prioritization, it is important to evaluate the effect of different workloads and hardware configurations on these questions. The importance of looking at different workloads is that an I/O bound workload may, for example, require a higher MPL, as disks need more simultaneous requests to perform efficiently. The importance of considering different hardware configurations is that a higher MPL may be required to achieve good throughput in a system with a large number of hardware resources, since more requests are needed to keep the many resources busy. We will therefore experiment with a wide range of hardware configurations and workloads, and two different DBMS.

Workload	Benchmark	Configuration	Database	Main memory	Bufferpool	CPU load	IO load
$W_{CPU-inventory}$	TPC-C	10 warehouses,	1GB	3GB	1GB	high	low
$W_{CPU-browsing}$	TPC-W Browsing	100 EBs, 10K items, 140K customers	300MB	3GB	500 MB	high	low
$W_{I/O-browsing}$	TPC-W Browsing	500 EBs, 10K items, 288K customers	2GB	512MB	100 MB	low	high
$W_{I/O-inventory}$	TPC-C	60 warehouses,	6GB	512MB	100MB	low	high
$W_{CPU+I/O-inventory}$	TPC-C	10 warehouses,	1GB	1GB	1GB	high	high
$W_{CPU-ordering}$	TPC-W Ordering	100 EBs, 10K items,140K customers	300MB	3GB	500MB	high	low

Table 1. *Description of the workloads used in the experiments.*

2.1. Experimental architectures

The DBMS we experiment with are IBM DB2 [1] version 8.1, and Shore [20]. Shore is a prototype storage manager with state-of-the-art transaction management, 2PL, and Aries-style recovery; we use it because we have the source code, enabling us to implement internal priorities. All of our external scheduling results are also corroborated using PostgreSQL [21] version 7.3, although we do not show these results here for lack of space.

The DBMS is running on a 2.4-GHz Pentium 4 running Linux 2.4.23. The buffer pool size and main memory size will depend on the workload (see Table 1). The machine is equipped with six 120GB IDE drives, one of which we use for the database log. The number of remaining IDE drives that we use for the data will depend on the particular experiment. The workload generator is run on a separate machine with the same specifications as the database server.

Setup	Workload	Number CPUs	Number disks	Isolation level
1	$W_{CPU-inventory}$	1	1	RR
2	$W_{CPU-inventory}$	2	1	RR
3	$W_{CPU-browsing}$	1	1	RR
4	$W_{CPU-browsing}$	2	1	RR
5	$W_{IO-inventory}$	1	1	RR
6	$W_{IO-inventory}$	1	2	RR
7	$W_{IO-inventory}$	1	3	RR
8	$W_{IO-inventory}$	1	4	RR
9	$W_{IO-browsing}$	1	1	RR
10	$W_{IO-browsing}$	1	4	RR
11	$W_{CPU+IO-inventory}$	1	1	RR
12	$W_{CPU+IO-inventory}$	2	4	RR
13	$W_{CPU-ordering}$	1	1	RR
14	$W_{CPU-ordering}$	1	1	UR
15	$W_{CPU-ordering}$	2	1	RR
16	$W_{CPU-ordering}$	2	1	UR
17	$W_{CPU-inventory}$	1	1	UR

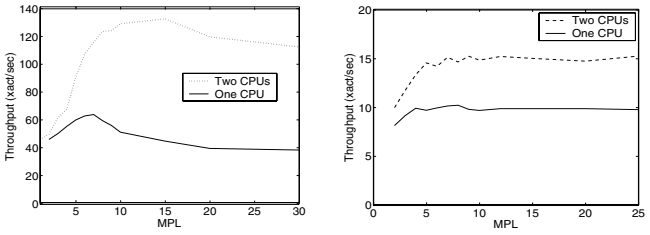
Table 2. *Definition of setups based on the workloads in Table 1.*

2.2. Experimental workloads and setups

When discussing the effect of the MPL it is important to consider a wide range of workloads. Unfortunately there are only a limited number of standard OLTP benchmarks which are both well-accepted and publicly available, in particular TPC-C [6] and TPC-W [7]. Fortunately, however, these two benchmarks can be used to create a much wider range of workloads by varying a large number of (i) hardware and (ii) benchmark configuration parameters. Table 1 describes the different workloads we create based on different configuration of the two benchmarks. The benchmark configuration parameters that we vary include: (a) the number of warehouses in TPC-C, (b) the size of the database in TPC-W (this includes both the number of items included in the database store and the number of "emulated browsers" (EBs) which affects the number of customers), and (c) the type of transaction mix used in TPC-W, particularly whether these are primarily "browsing" transactions or primarily "ordering" transactions. We run the workloads from Table 1 under different hardware configurations creating 17 different "Setups" as summarized in Table 2. The hardware parameters that we vary include: (a) the number of disks (1 – 6), (b) the number of CPUs (1 or 2), and (c) the main memory (ranging between 512 MB and 3 GB). We also vary the isolation level to create different levels of lock contention, starting with the default isolation level of 3 (corresponding to RR in DB2 – Repeatable Read), but also experimenting with lower isolation levels (UR – Uncommitted Read), leading to less lock contention. In all workloads, we hold the number of clients constant at 100.

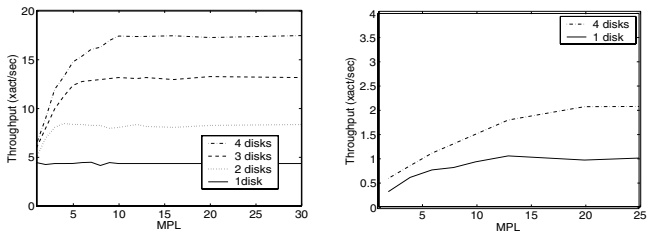
3. Feasibility of low MPL: Experimental study

In this section we ask how low can we make the MPL without causing deterioration in throughput and/or overall mean response time. The aim is to look at *low* values of the MPL and study their effect on throughput and then on mean response time using the experimental setups described in the previous section. (We will not be considering high values of the MPL, that are commonly looked at in studies dealing with overload and admission control.) We will be interested in identifying the *workload factors* that affect the answer to the question of "how low can one make the MPL." These results are summarized in Section 3.3.



(a) $W_{CPU-inventory}$ (b) $W_{CPU-browsing}$

Figure 2. *Effect of MPL on throughput in CPU bound workloads:(a) $W_{CPU-inventory}$ (Setups 1 and 2 of Table 2) and (b) $W_{CPU-browsing}$ (setups 3 and 4 of Table 2).*



(a) $W_{I/O-inventory}$ (b) $W_{I/O-browsing}$

Figure 3. *Effect of MPL on throughput in I/O bound workloads: (a) $W_{I/O-inventory}$ (setups 5–8 of Table 2) and (b) $W_{I/O(TPC-browsing}$ (setups 9 and 10 of Table 2).*

3.1. Effect on throughput

For CPU bound workloads

Figure 2 shows the effect of the MPL on the throughput under two CPU-bound workloads: $W_{CPU-inventory}$ and $W_{CPU-browsing}$. The two lines shown consider the case of 1 CPU versus 2 CPUs. In the single CPU case, under both workloads, the throughput reaches its maximum at an MPL of about 5. In the case of 2 CPUs, the maximum throughput is reached at around $MPL = 10$ in the case of workload $W_{CPU-inventory}$ and at around $MPL = 7$ in the case of workload $W_{CPU-browsing}$. Observe that a higher MPL is needed to reach maximum throughput in the case of 2 CPUs as compared with 1 CPU because more transactions are needed to saturate 2 CPUs. The fact that the $W_{CPU-inventory}$ requires a slightly higher MPL is likely due to the fact that the $W_{CPU-inventory}$ workload has some I/O components due to updates. The additional I/O component means that more transactions are needed to fully utilize the CPU, since some transactions are blocked on I/O to the database log. All these maximum throughput points are achieved at surprisingly low MPL values, considering the fact that both these workloads are intended to run with 100 clients according to the TPC specifications.

For I/O bound workloads

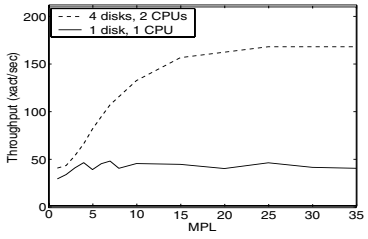
Figure 3 shows the effect of the MPL on the throughput under two I/O-bound workloads: $W_{I/O-inventory}$ and $W_{I/O-browsing}$. The lines shown consider different numbers of disks. The $W_{I/O-inventory}$ workload is a pure I/O-only workload, because of the larger database size. For this workload, the MPL point at which maximum throughput is reached is $MPL = 2$ for the case of 1 disk, $MPL = 5$ for the case of 2 disks, $MPL = 7$ for the case of 3 disks, and $MPL = 10$ for the case of 4 disks. Observe that the MPL needed to maximize throughput grows for systems with more disks, since more transactions are required to saturate more resources. Again, these numbers are extremely low considering the fact that the TPC specifications for this workload assumes 600 clients (we use 100 clients experimentally).

It is interesting to note that the the increase in MPL necessary to ensure x% of the maximum throughput is a somewhat *linear* function. We will give analytical validation for this observation in Section 4. Although it may appear problematic that the necessary MPL grows linearly with more disks, it is important to notice that systems with many disks also have a proportionately larger population of clients, hence an MPL that seems large may still be small in proportion to the client population.

For $W_{I/O-browsing}$, the MPL at which maximum throughput is reached is higher than for $W_{I/O-inventory}$ (about $MPL = 13$ for one disk and about $MPL = 20$ for four disks). The reason is that the size of this database is smaller than for the $W_{I/O-inventory}$ workload, thus resulting in a larger CPU component than in the purely I/O-based $W_{I/O-inventory}$. As explained in Section 3.1 the additional CPU component will add to the MPL needed. Still, it is surprising that an MPL of 20 suffices given that the TPC specifications for this workload assumes 500 clients (recall we use 100 clients experimentally).

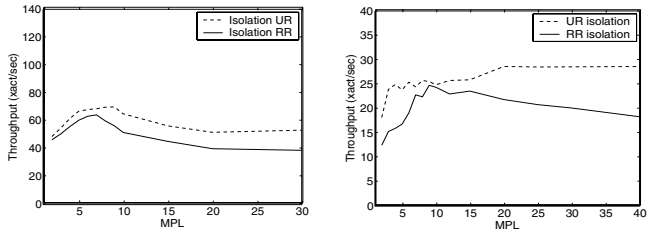
For "balanced" $CPU + IO$ workloads

Figure 4 considers workload $W_{CPU+I/O-inventory}$ which is balanced (equal) in its requirements of CPU and I/O (both resources are equally utilized). In the case of just 1 disk and 1 CPU, an MPL of 5 suffices to reach maximum throughput. Adding only disks to the hardware configuration changes this value only slightly, since the CPU bottleneck remains. Similarly, adding only CPUs changes the required MPL value only slightly, since now the workload becomes solely I/O bound. However if we add 4 disks and 2 CPUs (maintaining the initial balanced proportions of CPU and I/O), we find that the MPL needed to reach maximum throughput increases to around 20. This number is still low



$W_{CPU+I/O-inventory}$

Figure 4. *Effect of MPL on throughput in workload exhibiting both high I/O and CPU: $W_{CPU+I/O-inventory}$ (setups 11 and 12 of Table 2).*



(a) $W_{CPU-inventory}$ (b) $W_{CPU-ordering}$

Figure 5. *Effect of MPL on throughput in workloads with heavy locking: (a) $W_{CPU-inventory}$ (setups 1 and 17 of Table 2) and (b) $W_{CPU-ordering}$ (setups 15 and 16 of Table 2).*

in light of the fact that the TPC specified number of clients for this workload is 100.

In summary, the MPL required is largely proportional to the number of resources that are utilized in a system without an MPL. In a balanced workload the number of resources that are utilized will be high; hence the MPL is higher.

For Lock-bound workloads

Figure 5 illustrates the effect of increasing the locking needed by transactions (increasing the isolation level from UR to RR) on the MPL for workloads $W_{CPU-inventory}$ and $W_{CPU-ordering}$. While the MPL needed overall is always under 20, the basic trend is that increasing the amount of locking lowers the MPL. The reason is that when the amount of locking is high, throwing more transactions into the system doesn't increase the rate at which transactions complete, since they are all queueing. Beyond some point, increasing the number of transactions actually lowers the throughput, as seen in [5, 8, 12, 18].

3.2. Effect on response time

Section 3.1 showed that external scheduling with low MPL is feasible in that it doesn't cause a significant loss in throughput provided the MPL is not too low. Because we are working in a closed system, an immediate consequence of this fact is that the overall mean response time also does not suffer (see Little's Law [15]). However, this point is far less obvious for an *open system*, where response time is not inversely related to throughput. In this section we will investigate the effect of the MPL value on mean response time in great detail, starting with experimental work and then moving to queueing theoretic analysis.

Experimentally, we modify our experimental setup to an open system with Poisson arrivals. For the open system we find that for workloads based on TPC-C the response time is insensitive to the MPL value, provided it is at least 4. In the case of TPC-W based workloads, the MPL value needs to be at least 8, for a system utilization of 70%, and at least 15 if the system utilization increases to 90 in order to obtain close-to-optimal mean response times (when compared to the system without MPL).

The most important observation is that the degree to which the MPL affects the mean response time is dominated by the *variability of the workload*, rather than other factors such as the resource utilization. For example the workloads based on the TPC-W benchmark consistently require a higher MPL than the TPC-C based benchmarks, independent of whether they are CPU bound (e.g. $W_{CPU-browsing}$) or IO bound (e.g. $W_{IO-browsing}$). The reason is that the service demands of the transactions in the TPC-W benchmark are more variable than those in the TPC-C benchmark.

The above observation can be explained both intuitively as well as through queueing theory. Intuitively, a low MPL increases overall mean response time when short transactions (which in a standard, non-MPL system would have short response times) get stuck waiting behind very long transactions in the external queue (independently of whether the long transaction is IO-bound or CPU-bound). For this to happen the workload needs to exhibit high variability of the service requirements, i.e. the transaction mix must contain some transactions that are much longer than the average. From a theoretical perspective our external scheduling mechanism with MPL parameter can be viewed as a single unbounded First-in-first-out (FIFO) queue feeding into a Processor-Sharing (PS) server where only MPL jobs may share the PS server. A high MPL makes the system behave more like a PS server, while a low MPL makes it more similar to a FIFO server. In queueing theory it is well known that the mean response time at a FIFO server is directly affected by job size variability [13], while that of a PS server is insensitive to job size variability.

To get an idea of whether the levels of variability exhibited by the TPC-C and TPC-W benchmarks are representative, we obtain traces from one of the top-10 online retailers and from one of the top-10 auctioning sites in the

US for comparison. We compute the squared coefficient of variation (C^2), a standard statistical measure for variability, for both the traces and the benchmarks. We find that the C^2 values of the traces are in agreement with the TPC-C benchmark: In the TPC-C benchmark the C^2 value varies between 1.0 and 1.5 (depending on the setup), while the traces exhibit values for C^2 of around 2. The variability in the TPC-W benchmark is higher exhibiting C^2 values of 15.

3.3 Results: Factors influencing choice of MPL

Our aim in this section has been to determine how low we can feasibly make the MPL without noticeably hurting throughput and mean response time. We have seen, via a wide range of experimental workloads, that the answer to this question is strongly dominated by just a few key factors of the workload.

For throughput, what's important is the number of resources that the workload would utilize if run without an MPL. For example, if an IO-bound workload is run on a system with 4 disks, then a higher MPL is required than if the same workload is run on a system with only 1 disk.

With respect to not hurting overall mean response time, the dominant factor in lower-bounding the MPL is the variability in service demands of transactions. Workloads with more variable service demands require a higher MPL.

Importantly, we find that the question of how low one can feasibly make the MPL, both with respect to throughput and mean response time, is hardly affected by whether the workload is I/O bound, CPU bound, or lock bound. This is a surprising finding, and shows that the *number* of resources that must be utilized to keep throughput high is more important than the *type* of resources.

We note that the graphs shown in this section all assume a high offered load in terms of the transaction arrival rate, and as we have seen, it is quite feasible to make the MPL low with only small deterioration in throughput. When the offered load is low, the deterioration in throughput is even smaller, since the external queue is typically empty.

4. Finding the right MPL

The previous section demonstrates the general feasibility of external scheduling across a wide range of workloads. In all experiments an MPL of less than 20 suffices to achieve near optimal throughput and mean response time, while the number of clients is comparatively far higher than 20 (typically a hundred or several hundred).

However, the performance study in the previous section merely indicates the general existence of a good MPL value. The purpose of this section is to develop techniques for automatically tuning the MPL value to make the external scheduling approach viable in practice. We seek a method for identifying the lowest MPL value that limits throughput and response time penalties to some threshold specified by the DBA (e.g. "throughput should not drop by more than 5%").

Database workloads are complex, and exactly predicting throughput and response time numbers is generally not feasible. The key observation is that for us it suffices to predict how a given MPL *changes* throughput and mean response time relative to the optimal performance. The change in performance caused by an MPL value is strongly dominated by only a few parameters (as summarized in Section 3.3); the change in throughput is mostly affected by the number of parallel resources utilized inside the DBMS; the change in mean response time is mainly affected by the variability in the workload. In both cases *queueing-related effects* dominate, rather than other performance factors.

The above observations leads us to the idea of tuning the MPL through a feedback control loop augmented with queueing theoretic guidance. We start by developing queueing theoretic models and analysis to capture basic properties of the relationship between system throughput and response time and the MPL. We then use these models to predict a lower bound on the MPL that limits performance penalties to some specified threshold. While the analytically obtained MPL value might not be optimal, it provides the control loop with a good starting value. The control loop then optimizes this starting value in alternating observation and reaction phases. The observation phase collects data on the relevant performance metrics (throughput and mean response time) and the reaction phase updates the MPL accordingly, i.e. if the throughput is too low the MPL is increased and if it is too high the MPL is decreased.

In the remainder of this section we detail the above approach. We first explain the queueing theoretic methods for predicting the relationship between MPL and throughput (Section 4.1) and mean response time (Section 4.2). In Section 4.3, we show how this knowledge can be used in a feedback control loop to fine-tune the MPL parameter.

4.1. Queueing analysis of throughput vs. MPL

We start by creating a very simplistic model of the database internal resources as shown in Figure 6. [1] We model the MPL by using a "closed" system with a fixed (MPL) number of clients as represented in Figure 6. We assume that the service times of all devices are exponentially distributed with service rate proportional to their utilization in the unlimited system (with unbounded MPL).

The reason why such a simple model is sufficient is that we are only interested in achieved throughput relative to the

[1] Our current model includes only CPU and disk resources. We don't model memory (or bufferpool) as a separate resource since the time a transaction spends accessing memory is time it either occupies the CPU (memory hit) or utilizes a disk (memory miss) and is therefore accounted for.

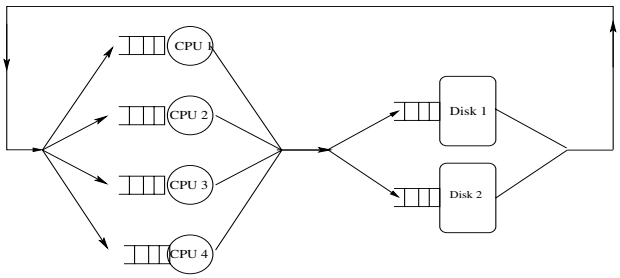


Figure 6. *Theoretical model representing the DBMS internals. This model provides us with a theoretical upper bound on the MPL needed to provide maximum throughput.*

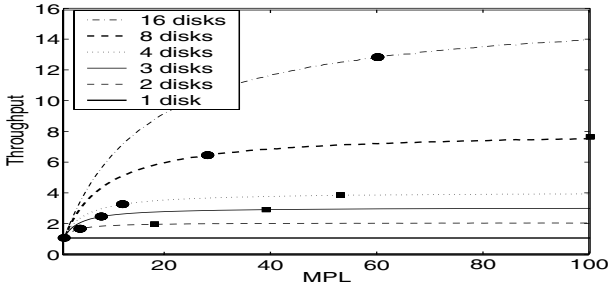


Figure 7. *Results of theoretical analysis showing the effect of the MPL on throughput as a function of the number of resources. The squares (circles) denote the minimum MPL that limits throughput loss to 5% (20%). Note that the set of circles form a perfectly straight line, as do the squares.*

optimal throughput. It is therefore not necessary to know the exact service demands at a device, just the relative proportions, since these will equally affect the throughput with and without MPL (e.g. a 5-times higher service demand will reduce throughput in both cases by a factor of 5). Moreover, in this type of queueing model the distribution of the service demand at the individual servers will not impact the throughput.

We analyze this "closed" system for different MPL values and and determine the achieved throughput. We compare the results to the maximum throughput for the system, until we find the lowest MPL value that leads to the desired throughput level (e.g. not more than 5% lower than the maximum throughput). Simple binary search can be used to make this process more efficient.

The MPL yielded by this analysis is in fact an upper bound on the actual MPL that we would get in experiments for two reasons: First, we purposely create the "worst-case" in our analytical model by assuming that all resources are equally utilized. This is realistic for the experimental setups that we consider, since we assume that the data is evenly striped over the disks and the CPU scheduler will ensure that on average all CPUs are equally utilized. For unbalanced workloads a smaller MPL might actually be feasible, and this could easily be integrated into the model. Second we do not allow for the fact that a client may be able to utilize two resources (e.g., two disks) at once.

To evaluate the usefulness of the model in predicting good MPL ranges we parameterize and evaluate the model based on the $W_{I/O-inventory}$ workload. For this workload there is almost no CPU usage, however the number of disks play an important role. In our experiments, we were able to experiment with up to 4 disks, as shown in Figure 3. However in analysis we can go much further. Figure 7 shows the results of the analysis with up to 16 disks. The first observation is that the results of the analysis for 1 to 4 disks look very similar to the actual experimental results from Figure 3. Next, we observe that the MPL required to reach near maximum throughput grows *linearly* with the number of disks: The minimum MPL that is sufficient to achieve 80% of the maximum throughput is marked with circles, and the minimum MPL that is sufficient to achieve 95% of the maximum throughput is marked with squares. Both the circles and the squares form *straight lines*. This matches the linear trend we also observed in experiments.

The take-away point is that simple queueing analysis, as we have done, captures the main trends of the throughput vs. MPL function well, and is a useful tool in obtaining an initial estimate of the MPL required to achieve the desired throughput. While we find that the current analysis is a very good predictor of our experimental results for the 4-disk system, it is certainly possible to refine the analytic queueing model further, or to integrate it with existing simulation tools for more realistic modeling of the hardware resources involved. However, such improvements are not crucial since the main purpose of the above model is merely to provide the controller with a good starting value, rather than a perfect prediction.

4.2. Queueing analysis of response time vs. MPL

Section 3 indicates that the effect of the MPL on the mean response time is dominated by the variability in the workload and hardly affected by other workload parameters such as the bottleneck resource or the level of lock contention. For workloads with little variability ($C^2 \approx 1$) MPL values around 4 are sufficient to achieve optimal mean response time, while more variable workloads ($C^2 \approx 15$) require an MPL of 8-15 (depending on system load). However, these particular results for the right choice of the MPL are hard to generalize, since they are based on only two benchmarks with two different levels of variability ($C^2 \approx 1$ and $C^2 \approx 15$). We therefore resort to analysis to obtain more general results.

From a theoretical perspective our external scheduling mechanism with MPL parameter can be viewed as a single unbounded First-in-first-out (FIFO) queue feeding into a Processor-Sharing (PS) server where only MPL jobs may share the PS server as illustrated in Figure 8. Note that this is not a poor approximation of our system in that, as we see in [22], Figure 8, the DBMS in many ways behaves like a PS system.

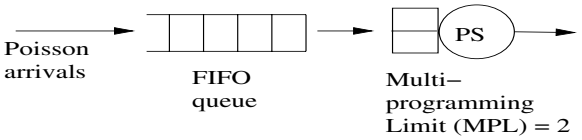


Figure 8. *Queueing network model of external scheduling mechanism with MPL = 2.*

To the best of our knowledge, there is no existing simple solution to our queueing network in Figure 8. Therefore, we derive the following solution approach: We start by modeling the job sizes (service requirements) by a 2-phase hyperexponential (H_2) distribution, with probability parameter p and rates μ_1 and μ_2, allowing us to arbitrarily vary the C^2 parameter. We can then represent the network in Figure 8 by an equivalent special "flexible multiserver queue" where the number of servers fluctuates between 1 and MPL as needed, and where the *sum* of the service rates at the multiple servers is always maintained constant and equal to that at the single PS server. The continuous-time Markov chain corresponding to the flexible multiserver queue is shown in Figure 9 for the case of an H_2 service time distribution (with parameters p, μ_1, and μ_2), arrival rate λ and MPL = 2. Note that we define the shorthand $q = 1 - p$. This Markov chain lends itself to Matrix-analytic analysis [14, 19], because of its repeating structure.

Figure 10 shows the results of evaluating the Markov chain in Figure 9. We find that for low C^2 values of 1 or 2, the mean response time is largely independent of the MPL value and equal to that for the pure PS system (with infinite MPL), assuming the MPL is at least 5. For higher C^2 values of 5–15, we find that the MPL depends on the load and needs to be at least 10 (for load of 0.7) or 30 (for load of 0.9) to ensure low mean response time (similar to PS).

4.3. A simple controller to find lowest feasible MPL

Next we explain how we use feedback control combined with queueing theory for tuning the MPL parameter.

When using feedback control for tuning parameters, the difficult part is choosing the right amount by which to adjust the parameter in each iteration: too small, conservative adjustments will lead to long convergence times, while too large adjustments can cause overshooting and oscillations. We circumvent the problem by using the queueing theoretic models from the previous subsections to "jump-start" the control-loop with a good, close-to-optimal starting value for the MPL. Initializing the control-loop with a close-to-optimal starting value provides fast convergence times, even given only small conservative constant adjustments.

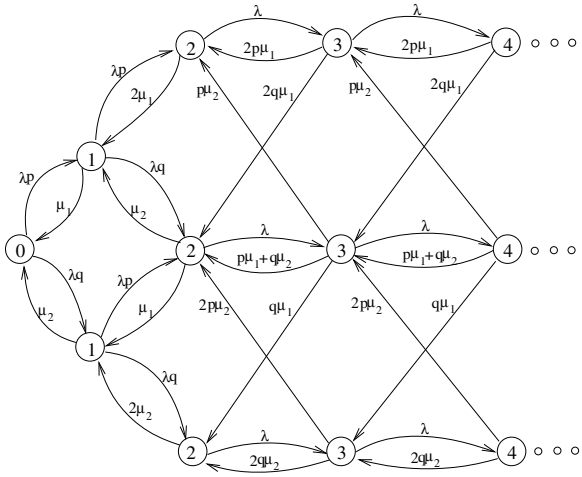


Figure 9. *Continuous-time Markov chain (CTMC) corresponding to the flexible multiserver queue representation of the queueing network in Figure 8. The two jobs in service may both have service rate μ_1 (top row), or may have rates μ_1 and μ_2 (middle row), or may both have service rates μ_2 (bottom row).*

A second critical factor in implementing the feedback based controller is the choice of the observation period. It needs to contain enough samples to provide a reliable estimate of mean response time and throughput. We determine the appropriate number of samples through the use of confidence intervals. For our workloads an observation period needs to span around 100 transactions to provide stable estimates. It is also important the observation period being studied does not have unusually low load, as this would cause low throughput independent of the current MPL used. Our controller takes the above two points into account by updating the MPL only after observation periods that contain a sufficient number of executed transactions and exhibit representative system loads.

We find in experiments that our queueing theoretically enhanced controller converges for all our experimental setups in less than 10 iterations to the desired MPL. While we find that using our simplistic control-loop is effective in determining the desired MPL, our approach could easily be extended to incorporate more complex control methods, e.g. following guidelines provided in [11]. This will be particularly useful for situations where queueing theoretical

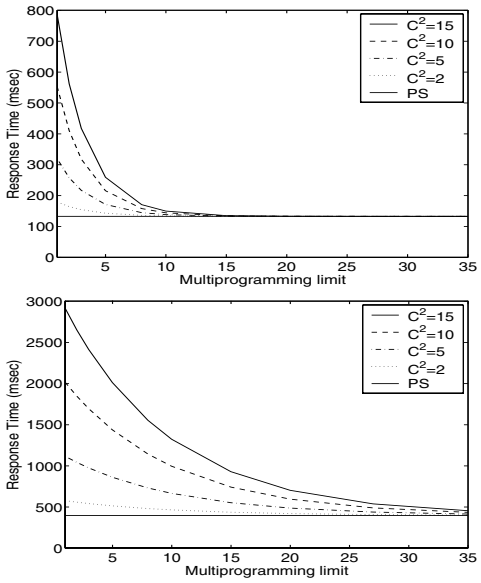


Figure 10. *Evaluation of CTMC for different C^2. The system load is 0.7 (top) and 0.9 (bottom).*

models are not precise enough in predicting good, close-to-optimal starting values for the controller.

5. External scheduling for Prioritization

Thus far we have presented an algorithm for finding a low MPL that doesn't hurt throughput or overall mean response time. The goal in keeping the MPL *low* is that a low MPL gives us control on the order in which transactions are scheduled, since we can *pick* the order in which transaction are dispatched from the external queue. Thus we are enabling certain transactions to run in isolation from others.

In this section, we apply our technique to the problem of differentiating between "high" and "low" priority transactions. Such a problem arises for example in the case of a database backend for a three-tiered e-commerce web site. A small fraction of the shoppers at the web site spend a large amount of money, whereas the remaining shoppers spend a small amount of money. It makes sense from an economic perspective to prioritize service to the "big spenders," providing them with lower mean response time.

We would like to offer high priority transactions low response times and low priority transactions higher response times. The lower the MPL that we use, the greater the differentiation we can create between high and low priority response times. At the same time we would like to keep the MPL high enough that throughput and overall mean response time are not hurt beyond a specified threshold. The technique presented in Section 4 allows us to achieve both of the above goals by specifying an exact MPL which will achieve the required throughput and overall mean response time, while being as low as possible, and hence providing maximal differentiation between high and low priority transactions.

In Section 5.1 we present results achieved via external prioritization, where, for each workload, the MPL is adjusted using the methods from Section 4. In Section 5.2 we discuss how one could alternatively implement prioritization internally to the DBMS by scheduling internal resources. Finally in Section 5.3, we compare the effectiveness of our external and internal approaches, and show that external scheduling, with the proper MPL, can be as effective as internal scheduling for our workloads.

5.1. Effectiveness of external prioritization

We start by implementing and studying the effectiveness of external prioritization. The algorithm that we use for prioritization is relatively simple. For any given MPL, we allow as many transactions into the system as allowed by the MPL, where the high-priority transactions are given first priority, and low-priority transactions are only chosen if there are no more high-priority transactions (see Figure 1). The MPL is held fixed during the entire experiment.

Note that this paper does not deal with how the transactions obtain their priority class. As stated earlier, we assume that the e-commerce vendor has reasons for choosing some transactions/clients to be higher or lower-priority. Experimentally, we handle this by simply at random assigning 10% of the transaction "high"-priority and the remainder "low"-priority.

We first consider the case where the MPL is adjusted to limit throughput loss to 5% (compared to the case where no external scheduling is used), see Figure 11(top), and then the case where the MPL is chosen to limit throughput loss to 20%, see Figure 11(bottom). For each of these two cases, we experiment with all 15 setups shown in Table 2. In each experiment we apply the external scheduling algorithm described in above and measure the mean response times for high and low priority transaction, in addition to the overall mean response time when no priorities are used.

We find that using external prioritization, in the case of 5% throughput loss (Figure 11(top)), high priority transactions perform 4.2 to 21.6 times better than low priority transactions with respect to mean response time. The average improvement of high priority transactions over low priority transactions is a factor of 12.1. The low priority transactions suffer only a little as compared to the case of no prioritization, by a factor ranging from 1.15 to 1.17, with an average suffering of 16 %. The above numbers are visible from the figure (or caption). Not visible from the figure is whether prioritization causes the overall mean response

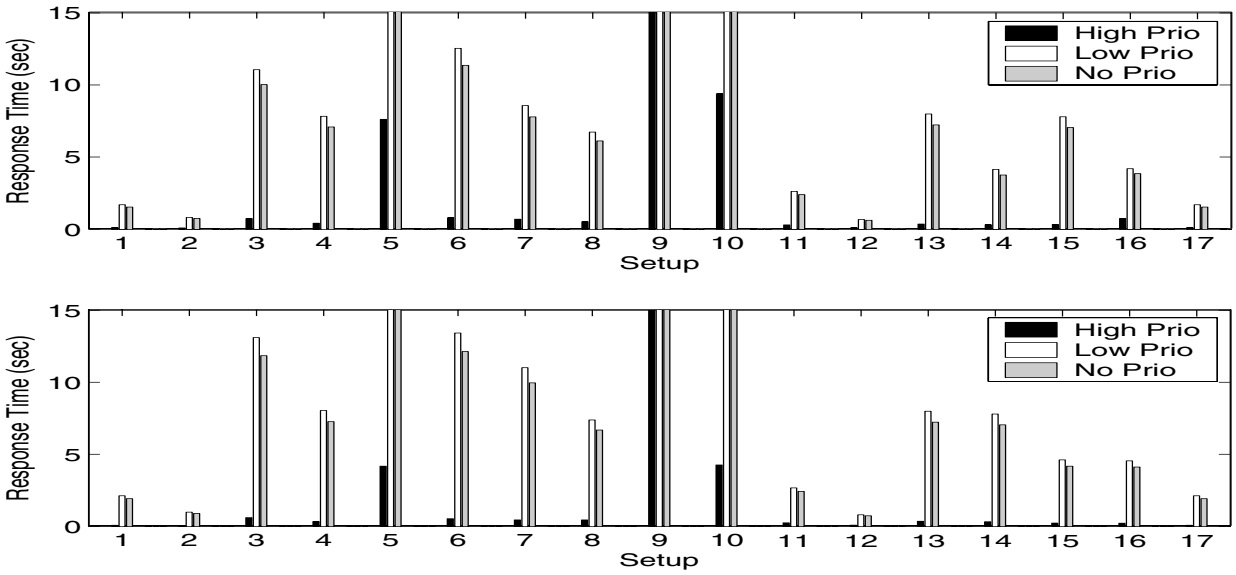


Figure 11. *Results of external scheduling algorithm. This figure shows the mean response times for high and low priority requests, as well as the case of no prioritization, for all 17 setups described in Table 2. In the* top *graph, the MPLs have been set to sacrifice a maximum of 5% throughput for each experiment. In the* bottom *graph, The MPLs are set to sacrifice a maximum of 20% throughput. Observe that workloads 5, 9, and 10 have been cut off. The values for these workloads in (top) are (7.6 sec, 76.864 sec), (26.2 sec, 111 sec), and (9.4 sec, 50.9 sec), respectively, and in (bottom) are (4.1 sec, 79.3 sec) (15 sec 112 sec) (4.2 sec and 51.9 sec) respectively.*

time to rise. It turns out that the overall mean response time is never hurt by more than 6% compared to the orginal system without external scheduling.

We find that using external prioritization, in the case of 20% throughput loss (Figure 11(bottom)), high priority transactions perform 7 to 24 times better than low priority transactions with respect to mean response time. The average improvement of high priority transactions over low priority transactions is a factor of 18. The low priority transactions suffer by a factor ranging from 1.35 to 1.39, as compared to the case of no prioritization, with an average suffering of 37%. The above numbers are visible from the figure (or caption). Not visible from the figure is whether prioritization causes the overall mean response time to rise. It turns out that the overall mean response time is never hurt by more than 25% compared to the orginal system without external scheduling. Observe that in the case of 20% throughput loss, the differentiation between high and low priority requests is more pronounced, since the MPL values are lower, but this comes at the cost of lower throughput and higher overall response times.

5.2. Implementation of internal scheduling

Scheduling the internals of the DBMS is obviously more involved than external scheduling. It is not even clear *which resource* should be prioritized: the CPU, the disk, the lock queues, etc. Once one resolves that first question, there is the follow-up question of *which algorithm* should we use to give priority to high-priority transactions, without extensively penalizing low priority transactions. Both questions are not obvious.

In a recent publication, [16], we address the first question of which resource should be prioritized via a detailed resource breakdown. We find that in OLTP workloads run on 2PL (2-phase locking) DBMS, transaction execution times are often dominated by lock waiting times, and hence prioritization of transactions is most effective when applied at the lock queue. We find that other workloads or DBMS lead to transaction execution times being dominated by CPU usage or I/O, and hence prioritization of transactions is most effective when applied at those other resources.

Having seen that it is not obvious which internal resource needs to be scheduled, we now turn to the particular 17 setups shown in Table 2. Some of these (e.g., setup 3 and 4) are CPU bound, while others (e.g., 1 and 2) are lock-bound, and still others are I/O bound (e.g. setup 5-10). In our experiments with internal scheduling we consider two particular setups: Setup 1 (Lock-bound) and Setup 3 (CPU-bound).

For setup 1, we implement the *Preempt-on-Wait* (POW) lock prioritization policy [17] in Shore [20]. In POW, high

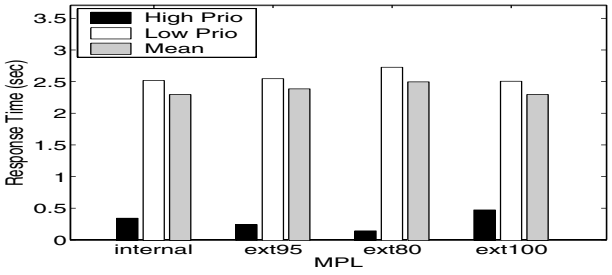


Figure 12. *Comparison of internal vs external prioritization for setup 1.*

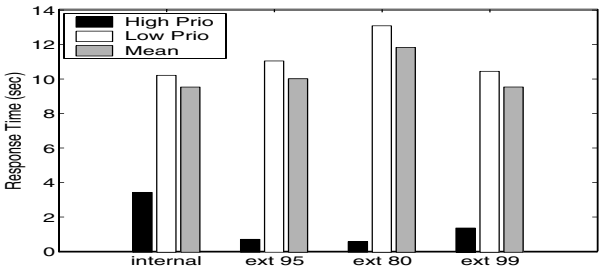


Figure 13. *Comparison of internal vs external prioritization for setup 3.*

priority transactions move ahead of low-priority transactions in the lock queue, and are allowed to even preempt a low-priority lock holder if that low-priority lock holder is waiting at another lock queue.

For setup 3, CPU prioritization is available in IBM DB2 through the DB2gov tool [3]. However, we find that we achieve better priority differentiation by "manually" setting the CPU scheduling priorities used by the Linux operating system. We use the `renice` command in Linux to set the CPU priority of a DB2 process executing a high priority transaction to -20 (the highest available CPU priority) and the CPU priority of a DB2 process executing a low priority transaction to 20 (the lowest available CPU priority).

In the next section we show the results for internal scheduling for these setups.

5.3. Internal prioritization results and comparison with external results

In this section we consider setup 1 and 3 from Table 2. For each setup, we compare the performance obtained via internal prioritization with that obtained via external prioritization. We consider 3 versions of external prioritization, the first involving 5% throughput loss, the second involving 20% throughput loss, and the third involving 0% throughput loss. Figure 12) shows the results for setup 1, and Figure 13 shows the results for setup 3.

For both setups, we find that with respect to differentiating between high and low priority transactions, external scheduling is nearly as effective as the internal scheduling algorithms that we looked at herein (for the case of zero throughput loss), and can even be more effective when the MPL is low (at the cost of a sacrifice in throughput). Looking at the suffering of the low priority transactions as compared to the overall mean response time, we find that external scheduling results in only negligibly more suffering for the low priority transactions, when compared with the internal scheduling algorithms herein. The penalty to the low priority transactions is minimized when the MPL is chosen so that no throughput is lost.

Because of the inherent difficulty in implementing internal scheduling, we were only able to provide numbers for setups 1 and 3 out of the 17 setups in Figure 2. However it is clear that for these two setups, external scheduling is a viable approach when compared with internal scheduling, and we hypothesize that external scheduling will compare favorably on the remaining setups as well, given the strong results shown for external scheduling in Figure 2.

We are not trying to say that external scheduling is always as effective as internal scheduling. Although the internal scheduling algorithms that we considered are quite advanced, there may be other internal scheduling algorithms which are superior to our external approach for certain workloads. Similarly, we are not trying to say that our proposed method for external scheduling is optimal. There may be many ways of further enhancing our external scheduler, for example by leveraging DBMS internal information on resource utilization, or information on resource demands of transactions. The point that we make in this paper is that external scheduling is a promising approach, when the MPL is adjusted appropriately.

6. Conclusion

This paper lays the experimental and theoretical groundwork for an exploration of the effectiveness of external scheduling of transactional workloads.

At the heart of our exploration is the question of *how* exactly should one limit the concurrent number of transactions allowed into the DBMS, i.e., the MPL (multi-programming limit). The obvious tradeoff is that one both wants the MPL to be low enough to create good prioritization differentiation and at the same time high enough so as not to limit throughput or create other undesirable effects like increasing overall mean response time.

Our work begins with an experimental study of how the MPL setting affects throughput and mean response. Our experiments include a vast array of 17 experimental setups (see Table 2), spanning a wide variety of hardware config-

urations and workloads, and two different DBMS (Shore, IBM DB2). We find that the choice of a good MPL is dominated by a few key factors. The dominant factor in lower-bounding the MPL with respect to minimizing throughput loss is the number of resources that the workload utilizes. The key factor in choosing an MPL so as not to hurt overall mean response time, is the variability in service demands of transactions. The fact of whether a workload is I/O bound, CPU bound, or lock bound is much less important in choosing a good MPL. Throughout we find that the values of MPL that are needed to ensure high throughput and low overall mean response time are in the lower range, in particular when compared with the typical number of users associated with the above experimental setup workloads.

The above experimental study encourages us to develop a tool for dynamically determining the MPL as a function of the workload and system configuration. The tool takes as input from the DBA the maximum acceptable loss in system throughput and increase in mean response time, and determines the lowest possible MPL that meets these conditions. The tool uses a combination of queueing theoretic models and a feedback based controller, based on our discovery of the dominant factors affecting throughput and overall mean response time.

Finally, we apply our tool for adjusting the MPL to the problem of providing priority differentiation. Given high and low priority transactions, we schedule the external queue based on these priorities (high priority transactions are allowed to move ahead of low priority transactions) and the current MPL. We experiment with different MPL values by configuring our tool with different thresholds for the maximum acceptable loss in system throughput and increase in mean response time. We find that the external scheduling mechanism is highly effective in providing prioritization differentiation. Specifically, we achieve a factor of 12 differentiation in mean response time between high and low priority transactions across our 17 experimental setups, if the MPL is adjusted to limit deterioration in throughput and mean response time to 5%. If we allow up to 20% deterioration in throughput and overall mean response time, we obtain a factor of 16 differentiation between high and low priority response times.

Lastly, to gauge the effectiveness of our external approach, we implement several internal prioritization mechanisms that schedule the lock resources and the CPU resources. We find that our external mechanism and internal mechanisms are comparable with respect to their effectiveness in providing priority differentiation for the workloads studied.

Our methods for dynamically adapting the MPL are very general. While we have applied them only to OLTP workloads in this paper, they are likely to apply to other workloads as well, and also to more general scheduling policies.

References

[1] DB2 product family. `http://www-3.ibm.com/software/data/db2/`.

[2] IBM DB2 query patroller adminsistration guide, ftp://ftp.software.ibm.com/ps/products/db2/info/vr7/pdf/letter/db2dwe70.pdf.

[3] DB2 Technical Support Knowledge Base. Chapter 28: Using the governor, http://www-3.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support /document.d2w/report?fn=db2v7d0frm3toc.htm.

[4] J. M. Blanquer, A. Batchelli, K. Schauser, and R. Wolski. Quorum: Flexible quality of service for internet services. In *Proceedings of NSDI '05*, 2005.

[5] M. J. Carey, S. Krishnamurthy, and M. Livny. Load control for locking: The 'half-and-half' approach. In *In ACM Symposium on Principles of Database Systems*, 1990.

[6] Transaction Processing Performance Council. TPC benchmark C. Number Revision 5.1.0, December 2002.

[7] Transaction Processing Performance Council. TPC benchmark W (web commerce). Number Revision 1.8, February 2002.

[8] H.U. Heiss and R. Wagner. Adaptive load control in transaction processing systems. In *Proceedings of Very Large Database Conference*, pages 47–54, 1991.

[9] Wei Jin, Jeffrey S. Chase, and Jasleen Kaur. Interposed proportional sharing for a storage service utility. In *Proceedings of* ACM SIGMETRICS '04, pages 37 – 48, 2004.

[10] A. Kamra, V. Misra, and E. Nahum. Yaksha: A controller for managing the performance of 3-tiered websites. In *Proceedings of IEEE International Workshop on Quality of Service (IWQoS 2004)*, 2004.

[11] C. Karamanolis, M. Karlsson, and X. Zhu. Designing controllable computer systems. In *Hot Topics in Operating Systems (HotOS'05)*, 2005.

[12] N. Katoh, T. Ibaraki, and T. Kameda. Cautious transaction schedulers with admission control. *ACM Trans. Database Syst.*, 10(2):205–229, 1985.

[13] Leonard Kleinrock. *Queueing Systems*, volume I. Theory. John Wiley & Sons, 1975.

[14] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, 1999.

[15] J. Little. A proof of the theorem $L = \lambda W$. *Operations Research*, 9:383 – 387, 1961.

[16] D. T. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *20th Int. Conference on Data Engineering (ICDE'2004)*, 2004.

[17] D. T. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Improving preemptive prioritization via statistical characterization of OLTP locking. In *21th International Conference on Data Engineering (ICDE'2005)*, 2005.

[18] A. Moenkeberg and G. Weikum. Performance evaluation of an adaptive and robust load control method for the avoidance of data-contention thrashing. In *Proceedings of Very Large Database Conference*, pages 432–443, 1992.

[19] M. F. Neuts. *Matrix-Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, 1981.

[20] University of Wisconsin. Shore - a high-performance, scalable, persistent object repository. `http://www.cs.wisc.edu/shore/`.

[21] PostgreSQL. `http://www.postgresql.org`.

[22] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum. Achieving class-based QoS for transactional workloads. In *22th International Conference on Data Engineering (ICDE'2006)*, 2006.