Performance Evaluation of an Adaptive and Robust Load Control Method for the Avoidance of Data–Contention Thrashing*

Axel Moenkeberg and Gerhard Weikum
ETH Zurich
Department of Computer Science
Information Systems – Databases
CH–8092 Zurich, Switzerland
E–mail: {moenkebe,weikum}@inf.ethz.ch

Abstract

Load control is necessary to prevent a database system from data–contention or memory–contention thrashing, caused by excessive lock conflicts or excessive buffer replacements that may occur due to temporary load peaks. The load control method that is adopted by virtually all commercial database systems is to limit the degree of multiprogramming (DMP), that is, the maximum number of transactions that are allowed to execute concurrently. Severe shortcomings of this DMP method are that it requires manual tuning by the human system administrator, and that it cannot react to dynamic changes of the transaction mix in the workload.

In this paper we present a performance evaluation of an adaptive, that is, self–tuning load control method for the avoidance of data–contention thrashing. The basic principle of this method is to monitor a data–contention performance metric called the conflict ratio, and to react to critical changes of the conflict ratio by temporarily suspending the admission of newly arriving transactions or by cancelling blocked transactions that block other transactions. In order to show the practical viability of our method, we have performed a performance evaluation based on page reference traces from the on–line transaction processing system of a large Swiss bank. The adaptive load control method copes well with the dynamic load fluctuations of this workload. It clearly outperforms the DMP method, since it adapts the system to the dynamic changes of the transaction mix whereas the DMP method uses a fixed DMP as a (bad) compromise for the entire duration of the experiment.

1 Introduction

1.1 The Problem

Performance tuning of database systems depends critically on the expertise and experience of system administrators and other human tuning experts who are responsible for the setting of system parameters. The purpose of such system parameters, or "tuning knobs", is to adapt the system's internalalgorithms and resource limits to the specific characteristics of an application workload. One of the most important parameters that virtually all commercial database systems provide is the degree of multiprogramming (*DMP*) in a multi–user environment. The DMP is a limit for the maximum number of transactions that are allowed to execute concurrently. During a load peak, when transactions arrive very frequently, only the specified maximum number of transactions are admitted for execution, and some transactions may be queued at the system entry before they acquire any resources.

The DMP serves to prevent the system from being temporarily overloaded, and aims to smooth out load peaks over longer periods of time while ensuring that the available resources (CPUs, disks, memory) are sufficiently well utilized. Without limiting the DMP, load peaks would lead to thrashing, that is, a sudden and sharp performance degradation, similar to virtual–memory thrashing in operating systems [De68]. Thrashing may occur for (at least) the following two reasons:

- memory contention, that is, excessive buffer replacements leading to disk I/Os that could be avoided if the working sets or "hot sets" of the executing transactions were held in memory, or
- data contention, that is, excessive lock conflicts leading to frequent and/or long blocking delays. In addition, depending on the reference characteristics of an application, the frequency of deadlocks may increase significantly.

This paper deals with the problem of data–contention thrashing (*DC thrashing*). Throughout this paper, we assume that two–phase locking is employed as the concurrency control protocol, as it is the case in virtually all commercial database systems. The DC thrashing phenomenon has been intensively discussed in various papers on the performance analysis of two–phase locking (e.g., [TGS85, ACL87, JTK89, TR91]). When DC thrashing occurs, more and more transactions become blocked so that the response time of transactions increases beyond acceptable values and essentially approaches infinity. In practice, DC thrashing is probably infrequent because the limitation of the DMP acts as a load control method. In addition, application

Proceedings of the 18th VLDB Conference
Vancouver, British Columbia, Canada 1992

* This work is partially supported by the UBILAB of the Union Bank of Switzerland (Schweizerische Bankgesellschaft).

programs are typically highly tuned in performance-critical applications (e.g., to reduce the probability of deadlock) and sometimes even sacrifice data consistency to avoid performance problems.

Unfortunately, the DMP method has two severe shortcomings as discussed in the following:

1. There are workloads that are very sensitive to changes of the DMP. If the DMP is set too high, then DC thrashing cannot be safely avoided. On the other hand, if the DMP is set too low, then transactions are often unnecessarily queued at the system entry and transaction response time becomes unacceptably high, too. Unfortunately, it is costly if not infeasible to determine the optimal DMP of an application by "trial-and-error" experiments, because one would need many operating hours of expensive production hardware to replay representative samples of the production workload. In practice, the setting of the DMP is probably based on a mix of simple experimentation and guesswork; or applications tend to be conservative in that their DMP is unnecessarily low and results in underutilized hardware.
2. In all commercial systems, the DMP is set "statically", that is, when the system is started up and configured according to the administrator's specification. Consequently, the DMP method cannot react to dynamic changes of the mix of transactions that constitute the current load. For example, a few heavy update transactions that occur once in a while may enforce a restrictive DMP, even if the workload consists of short and mostly read-only transactions most of the time. This problem may be alleviated by specifying DMP values for different overlapping classes of transaction types, which is supported by some TP monitors. However, this extended method makes the problem of finding the optimal combination of DMP values even trickier and ultimately unmanageable for most human administrators.

The above two problems are severe drawbacks of the DMP method for performance-critical database system applications such as on-line transaction processing (OLTP). Unfortunately, the problems will likely be further aggravated by the expected advances of database systems. Object-oriented and extended relational database systems are geared towards more complex transactions and more diverse workloads (e.g., mixing OLTP and decision-support queries). Because such systems will also speed up the application development process, the workload that is imposed on a system will change more rapidly and will become less predictable. In addition, with increasing interoperability across system boundaries, a significant fraction of the workload may become inherently unpredictable, and DMP settings that are based on the local load alone will be meaningless. The bottom line is that the DMP method is inappropriate as a load control method that can safely avoid DC thrashing in systems with complex, temporally changing, highly diverse, or simply unpredictable workloads.

1.2 Our Approach

This paper advocates an adaptive load control method, which has been proposed by the authors in [MW91] and is further elaborated and evaluated here. Unlike the DMP method, our adaptive method does not depend on any manual tuning directives; it is completely self-tuning in the sense that it adapts the DMP to the current workload dynamically and automatically. Its basic principle is to monitor a data-contention performance metric called the conflict ratio, and to react to critical changes of the conflict ratio by temporarily suspending the admission of newly arriving transactions or by cancelling blocked transactions that block other transactions. In addition, our method is able to exploit predictions for the length of a newly arriving transaction, that is, the number of locks that a transaction is going to request. However, our method does not depend on the availability of such advance knowledge; it is fairly robust without predictions or inaccurate predictions. More details of our load control method are presented in Section 2 and Section 3.

The performance of our load control method has been studied in [MW91] by means of simulation based on relatively simple, synthetic workloads. In this paper, we present a much more challenging performance evaluation based on real-life page reference traces. This evaluation is not only considered to be more significant than the synthetic-load simulations of [MW91] (and those of [CKL90, HW91], cf. Section 2.3), it also shows even bigger performance gains of the proposed adaptive load control method compared to the DMP method.

1.3 Contribution of the Paper

The focus of this paper is on the performance and the robustness of our load control algorithm. Robustness means that the algorithm has the following properties:

1. It does not only avoid DC thrashing and provide good average response time, but it also keeps the variance of the transaction response time as low as possible, so that the system performance perceived by the end-user becomes more predictable.
2. It does not require advance knowledge of the length of transactions, the access characteristics of transactions (e.g., specific access patterns), or the transaction mix (i.e., fractions of transaction types) of the near-future load. If such knowledge is available (e.g., based on average transaction lengths of the various transaction types), then it may be exploited for further performance improvements.
3. It can cope with highly diverse workloads, with a high variance of the transaction length. It is shown analytically in [TR91, Th91, Th92] that workloads with a high variance of the transaction length are much more susceptible to DC thrashing than more homogeneous workloads with the same average transaction length.
4. It can cope with load fluctuations where the transaction arrival rate changes rapidly due to bursts of transaction arrivals.
5. It can cope with dynamic changes of the transaction mix, that is, the relative frequency of the transaction types that constitute the workload. Such changes may occur independently of or in conjunction with dynamic changes of the overall arrival rate of transactions.

This paper provides evidence that our adaptive load control method does, in fact, satisfy the above requirements to a large extent. The paper presents a performance evaluation of our algorithm, based on real-life page reference traces that were gathered from the on-line transaction

Load control method	Performance metric	Transaction admission policy		Transaction cancellation policy	
		Trigger condition	Action	Trigger condition	Action
Adaptive method of [MW91], elaborated in this paper	Conflict ratio = ratio of the number of locks held by all transactions to the number of locks held by active transactions	Conflict ratio exceeds a critical threshold of 1.3 Conflict ratio drops below the critical threshold of 1.3	Suspend the admission of new transactions Admit one or more transactions of the BOT queue	Conflict ratio exceeds a critical threshold of 1.3	Cancel one or more transactions that are blocked and block other transactions
Half–and–half method [CKL90]	Fraction of blocked and mature transactions Fraction of active and mature transactions	Fraction of blocked and mature transactions exceeds 0.5 Fraction of active and mature transactions is below 0.5	Suspend the admission of new transactions Admit one or more transactions of the BOT queue	Fraction of blocked and mature transactions exceeds 0.5	Cancel one or more trans–actions that are blocked and block other transactions
Feedback method of [HW91]	Transaction throughput in time intervals of the recent past	Throughput has increased in last interval Throughput has decreased in last interval	Increase DMP Decrease DMP	none recommended	
Analytic model of [Th91]	Fraction of blocked transactions	Performance metric exceeds a critical threshold of 0.3	no details given	no details given	
Analytic model of [Th92]	Fraction of lock conflicts with blocked transactions = 1 – 1 / conflict ratio	Performance metric exceeds a critical threshold ϱ_{max} with $0.2 \leq \varrho_{max} \leq 0.3$ depending on transaction characteristics	no details given	no details given	
Wait–depth limitation [FR85, FRT91]	none	none		A transaction is blocked and blocks other transactions	Abort the trans–action
DMP method	none	Number of executing transactions is equal to the specified DMP limit	Suspend the admission of new transactions	none	

Figure 1: Comparison of DC Load Control Methods

processing system of a large Swiss bank. The diversity and variance of the transactions in these traces pose a stress test to our algorithm. Since the dynamics of real–life workloads cannot be modeled by stochastic distribution functions, it is crucial to perform this sort of experimental evaluation. We believe that the results from this performance evaluation are an important step beyond the simulations that were presented in the previous work on DC load control.

It is shown that our algorithm has robust behavior in the presence of the challenging workload properties discussed above. The algorithm not only avoids DC thrashing, but actually outperforms the DMP method since it adapts the system to the dynamic changes of the transaction mix. In addition to achieving good average response time, our method provides also acceptable variance of performance, as indicated by the 90th percentile of the transaction response time. These re–sults are achieved without assuming any advance knowledge of transaction lengths and access characteristics. The paper also discusses the performance impact of such knowledge, including the impact of inaccurate predictions.

The rest of the paper is organized as follows. Section 2 discusses previous work on data–contention load control methods, including our own adaptive method [MW91], which is further elaborated here. Section 3 discusses various options how to exploit predictions of the transaction lengths and access characteristics. Section 4 presents the set–up of our performance experiments and the workload that drove the experiments. Section 5 presents the experimental results, and discusses our major findings. The paper is concluded with an outlook on the next steps in our work towards a practically viable load control method.

2 Previous Work

The problem of DC thrashing and the need for load control were pointed out in the literature on performance analysis of two–phase locking (e.g., [TGS85, ACL87, JTK89, TR91]), and has been discussed in some textbooks and tutorials (e.g., [BHG87, Hae87, Sha92]). However, the performance analysis work is not constructive as it does not make any algorithmic proposals how to avoid DC thrashing. More recently, a number of researchers, including ourselves, have proposed adaptive load control methods that aim to prevent systems from DC thrashing while ensuring that the available resources are sufficiently well utilized [CKL90, MW91, HW91]. In addition, some papers on performance analysis of two–phase locking have provided an analytic explanation of the DC thrashing phenomenon, and have contributed elements of load control methods without giving complete algorithms [FR85, FRT91, Th91, Th92]. The basic idea of all these approaches is to monitor some performance metric that indicates the current degree of data contention in the system, and to react to critical changes of the metric. The various approaches differ in their underlying performance metric and in their possible reactions to critical changes. These differences are summarized in Figure 1, and are further discussed in the following.

2.1 Performance Metrics

Our own approach [MW91] is based on the conflict ratio metric. The conflict ratio at a given point of time is the ratio of the total number of locks that are currently held by all transactions in the system and the total number of locks held by active, i.e., non–blocked transactions.[1] The best possible value of the conflict ratio is 1.0, that is, when no transaction is blocked. The conflict ratio increases with the fraction of blocked transactions, where the impact of a blocked transaction increases with the number of locks that it has already acquired. Through extensive simulation experiments we found that DC thrashing is about to occur when the conflict ratio exceeds a critical threshold of 1.3. The point about this threshold is that it is essentially workload–independent; that is, a conflict ratio of 1.3 or higher indicates DC thrashing, regardless of the length and the access characteristics of the transactions in the system. In particular, the conflict ratio and its critical threshold are independent of the fractions of shared and exclusive locks.

The existence of such a universal constant was suggested already by the seminal work of Tay [TGS85], and its exact value was further investigated by means of analytic modeling in the recent work of Thomasian [TR91, Th91, Th92]. The results presented in [Th92] do in fact provide a thorough analytical justification for using the conflict ratio as a performance metric for DC load control.

Alternative metrics that have been proposed in [CKL90, MW91, Th91, Th92] are: 1) the fraction of blocked transactions in the system (called the "transaction metric" in [MW91]), and 2) the fraction of lock conflicts with blocked transactions. The fraction of lock conflicts with blocked transactions is equivalent to the ratio of the total number of locks held by blocked transactions and the total number of locks held by all transactions; so it is actually the same as 1.0 minus the inverse of the conflict ratio.

The fraction of blocked transactions is essentially used as the performance metric of the "half–and–half" load control method [CKL90]. However, in this approach, "non–mature" transactions, which have not yet acquired at least 25% of their locks, are discounted. Note that this notion of mature vs. non–mature transactions requires some advance prediction of the number of locks that a transaction is going to acquire. The degree of data contention is considered critical when the fraction of blocked and mature transactions exceeds 0.5 (hence the name of the method). The model of [Th91, Th92], on the other hand, yields a value of approximately 0.3 for the fraction of blocked transactions at the DC thrashing point, taking into account both mature and non–mature transactions. It seems, however, that the critical value of this blocked–transactions metric is more sensitive to the characteristics of the workload and therefore less appropriate as a load control threshold, compared to the critical threshold of the conflict ratio (cf. [Th92]). The instability of the critical threshold of the blocked–transactions metric is probably one reason for distinguishing mature vs. non–mature transactions in the half–and–half method.

An approach that uses transaction throughput as its performance metric is the feedback method proposed in [HW91, Hei91]. The feedback method continuously measures the transaction throughput over fixed–size time intervals, and it attempts to track the optimal DMP by reacting to increases or decreases of the transaction throughput. If the throughput in the last measurement interval has increased (compared to the interval before), the DMP is increased; if the throughput has decreased, the DMP is decreased, too. In both cases, the amount by which the DMP is adjusted depends on the gradient of the observed throughput curve (i.e., the transaction throughput as a function of the DMP values of the past intervals), using mathematical approximation techniques.

A problem with the feedback method is that it crucially depends on a proper choice of the size of the measurement intervals. If the intervals are too big, then the feedback method is not reactive enough in the presence of rapid load fluctuations; if the intervals are too short, then the algorithm may overreact to stochastic noise. It is not clear to what extent the size of the measurement intervals needs to be tuned to the workload at hand. It still needs to be shown that the method does not need any manual tuning hints, and that it can cope also with highly heterogeneous workloads (e.g., a mix of frequently arriving, short transactions and infrequent, long transactions).

2.2 Reaction to Overload

The policy how to react to critical changes of the underlying DC performance metric can be decomposed into a transaction admission policy and a transaction cancellation policy, as illustrated in Figure 2 for our algorithm. The transaction admission control is invoked when a new transaction arrives and issues a BOT (i.e., Begin–Of–Transaction) request. The transaction is either admitted immediately when the DC performance metric is in the uncritical range, or is held in a BOT queue at the system entry when the performance metric is above the critical threshold. In the latter case, the transaction will be reconsidered for admission at a later point of time.

2.2.1 Transaction Admission

Several policies are conceivable as to when the transactions in the BOT queue are reconsidered for admission. Our load control method reconsiders non–admitted transactions at each EOT (End–Of–Transaction), that is, when a transaction commits and leaves the system, or when a transaction is aborted (due to deadlock or cancellation) and reenters the BOT queue. In either of these cases, the conflict ratio, that is, our performance metric, may have decreased, and if it has dropped below the critical threshold, then one or more of the transactions in the BOT queue are admitted. The basic version of our method simply admits all transactions of the BOT queue whenever it decides to admit one of them. However, transactions that were aborted due to deadlock and request to be restarted are delayed (see Subsection 2.2.2). If our load control method is provided with advance knowledge about transaction lengths, then it may decide to admit only a subset of the transactions that are waiting in the BOT queue (see Section 3).

1. In [MW91], we referred to this metric as the *conflict rate*. The term conflict *ratio* seems more appropriate, since a *rate* usually refers to the frequency of some event type per time unit. The conflict ratio always refers to the current state of the system.

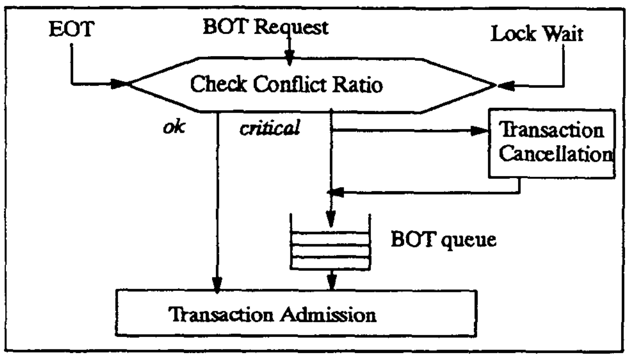

Figure 2: Transaction Admission and Cancellation in the Adaptive Load Control Method

The half–and–half method, on the other hand, reconsiders transactions in the BOT queue at each lock request. If the fraction of active and mature transactions is above 50%, then all transactions in the BOT queue are admitted. In addition, when a transaction commits, either the first transaction in the BOT queue or, if the BOT queue is empty, the next arriving transaction will be admitted, regardless of the performance metric. Note that, once the BOT queue is non–empty, the half–and–half method checks its performance metric much more frequently than our method, since it reconsiders transaction admissions at each lock request. This seems to be natural as the state of transactions may change from non–mature to mature upon a lock request, thus changing the value of the performance metric for admission control. However, it also seems to make the method overreactive to rapidly changing workload characteristics, whereas our algorithm tends to be more robust.

The feedback method [HW91] does not have an explicit admission policy. Transactions are admitted whenever the current optimum of the DMP that is determined by the method is higher than the number of transactions that are currently executing.

2.2.2 Transaction Cancellation

The impact of an admitted transaction on the degree of data contention becomes effective only some time after the admission, namely when the transaction has acquired a significant fraction of its locks. Therefore, transaction admission control alone may not be sufficient for load control. In addition, the load control method may consider cancelling one or more transactions when the performance metric becomes critical. Here, cancellation means that a transaction is aborted and restarted, subject to transaction admission control and possibly after some delay interval.

Our load control algorithm cancels one or more transactions whenever its performance metric, the conflict ratio, is above the critical threshold. This condition is checked at each lock wait, i.e., lock requests that are not granted and lead to the blocking of a transaction. The algorithm cancels as many transactions as necessary to bring the conflict ratio below the threshold. However, it will cancel only transactions that are blocked and do in turn block other transactions.

Cancellation victims are picked in ascending order of the product: number of locks held by the transaction * number of aborts and restarts that the transaction experienced so far. Picking the transaction with the fewest locks has been found to be a good policy for selecting deadlock victims in [ACM87], and it is extended here to prevent transactions from starvation. The testbed on which we performed our performance evaluations (see Section 4) employs the same policy for deadlock resolution, too. A deadlock victim is restarted only after all other transactions in its wait–for–graph cycle are committed; this policy has been called Restart Waiting in [TR90]. Cancellation victims are restarted (i.e., placed in the BOT queue) without any delay. Both cancellation and deadlock victims are prioritized over newly arriving transactions by the admission policy.

So our load control method cancels only transactions that do not make progress and, in addition, prevent other transactions from making progress. This cancellation policy has been proposed in [FR85, FRT91], where it is used unconditionally, i.e., independently of the degree of data contention in the system, in order to limit the length of lock wait queues. [FRT91] also proposes a criterion for carefully selecting cancellation victims, based on the number of locks that are currently held by the candidate transactions. Note however, that this unconditional cancellation policy may incur unnecessary work by aborting and restarting a transaction even if the system is far below the DC thrashing point and transactions make reasonable progress.

The half–and–half method essentially uses the same cancellation policy as our algorithm. When the fraction of blocked and mature transactions is higher than 0.5, then transactions are cancelled until this metric drops below 0.5. Compared to our method, the half–and–half method further restricts the selection of cancellation victims to mature transactions. This restriction is in line with the underlying performance metric of the method, since only lock releases by mature transactions will affect the method's perception of data contention. This observation again conveys the main problem of the half–and–half method: in order to improve the stability of the blocked–transactions metric as a data–contention indicator, the notion of mature vs. non–mature transactions is introduced. However, basing all decisions only on mature transactions seems to make the algorithm less robust than our method, and it requires advance estimates of the length of transactions.

The feedback method [HW91] discusses cancellation policies only briefly. It concludes that cancellation should not be employed, or only as a last resort. Consequently, when the optimal DMP is decreased at some point of time and the current number of executing transactions is δ transactions above the new optimum, the effective DMP will decrease only after the commit of δ transactions. According to [HW91], this delay has a smoothing effect on the variation of the DMP; on the other hand, it seems to make the algorithm less reactive to rapidly emerging overload.

2.3 Experimental Results

The half–and–half method and our load control method have been evaluated based on simulations driven by synthetic workloads with relatively few different transaction types [CKL90, MW91]. Both methods performed reasonably well in that they safely avoided DC thrashing and achieved transaction response time results that were not much worse and sometimes even better than those of a manually tuned system using the DMP method.

Both evaluations mostly concentrated on the case of "steady overload", that is, a transaction arrival rate that is higher than the system's sustained throughput (for the given average transaction length). Only preliminary results were presented on the impact of dynamic load fluctuations, where the arrival rate may increase temporarily due to load bursts or the transaction mix may change dynamically. In a synthetic load-peak experiment, where the transaction mix changed periodically, our method outperformed the half-and-half method [MW91]. Similar simulation experiments are described in [HW91] for the feedback method, without comparison to other methods. However, none of these simulation studies captures the diversity and fluctuations of real-life workloads, which is the real touchstone for adaptive load control methods.

3 Exploiting Advance Knowledge of Transaction Properties

In this section we discuss how the effectivity of our load control method can be potentially further improved by exploiting advance knowledge of the access characteristics of transactions. Suppose that the conflict ratio of a database system is slightly below the critical threshold, and that a long update transaction arrives and requests to be admitted. The admission control would usually admit the transaction. However, if it knew in advance that the transaction is going to acquire a large number of locks, then it may infer that the conflict ratio will most likely increase beyond the critical threshold due to the increased probability of lock conflicts. In this case, the admission control should better decide to postpone the admission of the transaction until after the commit of some (or, in the extreme case, even all) of the currently executing transactions, so that the long update transaction will not cause DC thrashing.

More generally, the advance knowledge of certain properties of a transaction can be used to estimate the effect on the conflict ratio if the transaction were admitted, and the admission decisions could be based on the estimated rather than the currently observed value of the conflict ratio. One could even estimate the effect of the expected lock requests of the transactions that are already executing, and may suspend the admission of all new transactions if the conflict ratio is expected to become critical in the near future.

Transaction properties that are useful to know for this sort of intelligent admission control are:

1. the length of a transaction, that is, the number of locks that it is going to request,
2. the CPU time and the number of disk I/Os for each transaction step, so that one could estimate the duration for which a lock is held, at least in a (resource- and data-) contention-free system,
3. the transaction's fraction of write accesses, that is, exclusive locks that it is going to request,
4. the reference pattern of the transaction (e.g., index traversal, sequential scan over a certain set of pages, etc.).

In general, it is unrealistic to assume that all this information is available on-line and accurately. However, because most production systems run "canned transactions" where transactions are classified into transaction types that correspond to pre-compiled transaction programs, at least some of the desired information may be available in the form of average values for individual transaction types. In the following, we assume that the average length of each transaction type is known in advance. Moreover, in many systems, these statistics are available for individual database partitions, i.e., tables or tablespaces/indexspaces in a relational system, areas in a Codasyl system, or "object clusters" in an object-oriented database system. So we assume that the average number of locks is known for each pair of database partition and transaction type.

One may object that average values are not meaningful if the variance of the transaction length is high. However, while this is true for some applications, other applications exhibit relatively low variance of the length of most transaction types. The challenge for the adaptive load control is to exploit this knowledge for the "predictable" transaction types, while avoiding that inaccurate predictions for high-variance transaction types affect the performance in an adverse way. Note that advance knowledge along these lines has been shown to be beneficial for affinity-based transaction routing methods in shared-disk systems [Ra89, Reu86, YBL88]. The exploitation of advance knowledge has also been proposed for conflict-avoiding concurrency control methods [BSR80, Bu89].

In the following, we show how the knowledge of the length l_i of a transaction T_i can be used to estimate the conflict ratio if the transaction were admitted for execution. We will use only simple heuristic formulas that can be evaluated at run-time with little overhead. Suppose that the database consists of D lock granules such as pages, the current number of locks held by all transactions in the system is L, and the current number of locks held by active, i.e., non-blocked transactions is A. Thus, the current conflict ratio is L/A. Further suppose, for now, that all locks are exclusive locks, and that lock granules are accessed according to a uniform distribution. Finally, let n be the number of currently executing transactions in the system. Then, the probability that a new transaction T_{n+1} will become blocked if it were executed is:

$$P\,[\,T_{n+1}\,\textit{is blocked}\,] \;=\; 1 - \prod_{k=0}^{l_{n+1}-1} \frac{D-L-k}{D-k} \qquad (1)$$

This result can be used to adjust the conflict ratio by the following heuristic formula, which simply adds a weighted number of locks to the number of locks held by active transactions:

$$CR' = \frac{L + l_{n+1}}{A + (1 - P[\,T_{n+1}\,\textit{is blocked}\,]) * l_{n+1}} \qquad (2)$$

This rough estimate does not yet take into account the fact that the currently executing transactions will also request further locks, which increase the conflict potential for the new transaction and may also lead to blocking among the current transactions. Let l_i' be the number of locks that are still to be requested by transaction T_i $(1 \le i \le n)$. A simple heuristics to accomodate this conflict potential is to view all these residual locks as a single fictitious transaction T^0 of length $\sum_{i=1}^{n} l_i'$. Then the estimated conflict ratio CR' can be adjusted by computing the blocking probability of T^0 and by setting:

$$CR'' = \frac{L + l_{n+1} + \sum_{i=1}^{n} l_i'}{A + (1 - P[T_{n+1} \text{ is blocked}]) \cdot l_{n+1} + (1 - P[T^0 \text{ is blocked}]) \cdot \sum_{i=1}^{n} l_i'} \tag{3}$$

This formula yields an upper bound for the conflict ratio that one would estimate if the blocking probabilities of the n current transactions were computed and incorporated individually.[2] The formula can be extended analogously, if one wants to estimate the effect of admitting more than one transaction at the same time. Likewise, the formula can be refined so as to take into account knowledge of the number of lock requests for each pair of transaction type and database partition. Suppose that the database consists of m partitions of size D_j $(1 \leq j \leq m)$. Let L_j, A_j, and l_{ij} denote the number of locks on partition j held by all transactions, all active transactions, and requested by transaction T_i, respectively. Then the probability that a newly admitted transaction T_{n+1} will become blocked is estimated by the following formula (4); this formula can be directly substituted into formula (3) to estimate the conflict ratio, as in the case of a non-partitioned database discussed before.

$$P[\,T_{n+1} \text{ is blocked}\,] = 1 - \prod_{j=1}^{m} \prod_{k=0}^{l_{n+1,j}-1} \frac{D_j - L_j - k}{D_j - k} \tag{4}$$

Now let us relax the conditions that all locks are exclusive and that lock granules are accessed uniformly. Suppose we know the fraction S of shared locks, averaged over all transaction types. Further suppose that the skew in the distribution of the access frequency of lock granules is specified by two numbers a and b $(0 < a,\ b < 1)$ where b is the fraction of the database that is accessed by a fraction a of the data references. For example, values $a=0.8$ and $b=0.2$ denote a Zipf-like access skew where 80% of the accesses refer to 20% of the data. The size of the database that is used in the predictions of the conflict ratio can then be "normalized" as follows:

$$D' = \frac{D}{1 - S^2}$$

$$D'' = \frac{D'}{1 + (a-b)^2/b(1-b)} = \frac{D}{(1 + (a-b)^2/b(1-b)) * (1 - S^2)} \tag{5}$$

It is shown in [TGS85] that, under a number of simplifying assumptions, a workload with a fraction S of shared locks and an $a-b$ access skew on a database of size D has the same degree of data contention as a workload with exclusive locks only and uniformly distributed accesses on a "normalized" database of size D''. For example, with $S=0.7$, $a=0.8$, and $b=0.2$, a database of size D = 1'000'000 pages is equivalent to a "normalized" database of size $D'' \approx$ 600'000 pages. Our adaptive load control method uses the above formulas to estimate the effect of a transaction admission on the conflict ratio. The amount of advance knowledge that is exploited gives rise to the following three different configurations of our algorithm.

1. *Ignorant*: This is the basic version of the load control algorithm, in which no assumptions are made about the availability of advance knowledge.
2. *Smart*: The expected change of the conflict ratio is estimated based on the average length of transaction types (formulas (1) and (3)).
3. *Intelligent*: The expected change of the conflict ratio is estimated based on the average number of lock requests for each pair of transaction type and database partition (formulas (4) and (3)).

In addition to providing a better basis for making admission decisions, the estimates of the conflict ratio can also be used for managing the BOT queue in a more aggressive way. Usually, the queueing discipline of the BOT queue is strict FIFO, with the exception of aborted and restarted transactions which are always placed at the front of the queue. With the ability to estimate the expected change of the conflict ratio that would be caused by the admission of one or more transactions, the admission control selects a maximum set of transactions from the front of the BOT queue such that the estimated conflict ratio will still remain below the critical threshold. Alternatively, a non-FIFO queueing discipline may be employed such that an arbitrary maximum subset of transactions in the BOT queue can be selected for admission. In this case, the admission control still processes the BOT queue in FIFO order. However, when a transaction is rejected because the admission of this transaction would increase the estimated conflict ratio beyond the critical threshold, the admission control will continue looking for further admission candidates with a lower conflict potential. This non-FIFO policy bears the risk of transaction starvation, that is, a transaction keeps being passed by less DC-critical transactions and remains in the BOT queue virtually forever. To reduce the probability of starvation, the algorithm artificially shrinks the transactions in the BOT queue by decreasing (i.e., dividing by two) the predicted length of a transaction each time a transaction is passed by another transaction. A better solution would be to introduce explicit priorities to reflect response time constraints of the individual transaction types, but this is beyond the scope of this paper.

4 Set-up of Performance Experiments

This section presents the organization of our performance experiments. Subsection 4.1 describes the testbed on which the experiments were performed, and Subsection 4.2 describes the real-life workload that drove the experiments.

4.1 System Configuration

We have implemented the described variants of the adaptive load control method as a component of the COMFORT prototype [Wei90]. COMFORT is an experimental self-tuning storage server for complex objects, which currently consists of a multi-disk file system providing coarse-grained file striping and automatic disk load balancing, a sophisticated buffer manager with a spectrum of adaptive DBMIN-like policies, a transaction manager using two-phase locking, and the load control component. This prototype served as the testbed for our performance evaluation. Since the purpose of the experiments was to evaluate the DC load control method, all other components were configured in a "vanilla" way. That is, files were not striped but

2. Even then, however, the formula would be far from being accurate, because it does not capture the fact that the residual steps of the n transactions execute concurrently, may have different durations, etc.

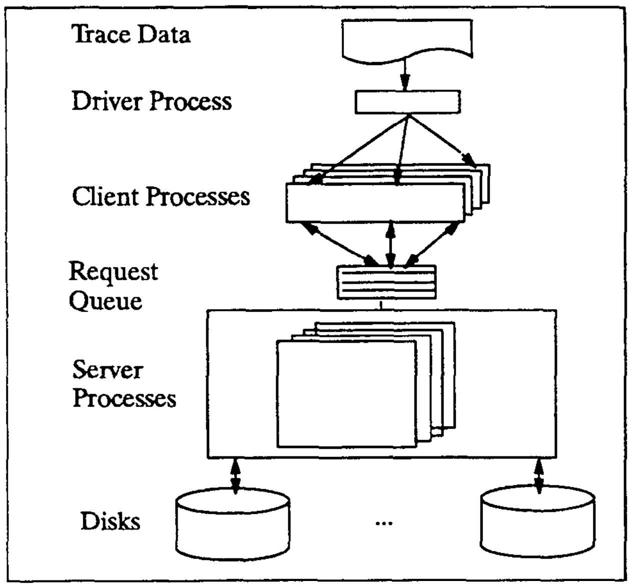

Figure 3: Architecture of the Performance Testbed

simply placed across (raw partitions of) eight disks with coarse load balancing by hand, and the buffer manager was configured to use global LRU for a page buffer of 1000 pages. The lock granule was set to pages, since the trace data that drove the experiments consist of pages references (see Section 4.2).

The experiments were run stand–alone on a Sequent Symmetry S81 shared–memory multiprocessor with 12 processors (Intel 80386), 8 disks (Fujitsu M2344K), 2 dual–channel controllers, and 80 MB memory. COMFORT is designed for a client–server architecture with multiple server processes that share the load and keep all control blocks in shared memory, as illustrated in Figure 3. For better control of the experiment, the client processes were run on the same machine as the server processes. In this experiment, clients were mere stubs from which the transactions were invoked. The clients did not perform any application processing; they merely placed page access requests (read or write), BOT requests, and Commit requests in a request queue serviced by the server processes. The clients were fed by a separate driver process which read the trace data and assigned transactions to clients. The experiments were performed with 50 client processes and 8 server processes. Server processes were prioritized and bound to real processors.

4.2 Workload Description

Our performance evaluation was driven by page reference traces that were gathered from the on–line transaction processing system of the Union Bank of Switzerland, located at Zurich. The traces were collected during one hour of high activity (10 AM to 11 AM). In this hour, about 46'000 transactions were processed against a Codasyl database consisting of more than 150 areas with a total size of approximately 23 GBytes.

For the performance experiments described in this paper, we selected a subset of these transactions, namely all transactions that belong to the local stock market application. This subset contained 2163 transactions, that is, about 5 percent of the total number of transactions. However, in terms of the number of page references, the stock market transactions represent the heaviest application with a total of 36'621 page references, which is about 6.6 percent of the overall load. These page references were distributed across 10 areas with a total size of about 730'000 pages. The stock market transactions exhibited an 80–40 access skew; that is, 80 percent of the page references referred to 40 percent of the potentially accessible pages. The average fraction of write accesses was 21 percent. These figures translated into a "normalized" database size of approximately 1.1 million pages (see Section 3, formula (5)). The 80–40 access skew was observed for the entire database (i.e., the 10 relevant areas) and within individual areas. There were no obvious individual hot–spot pages.

The selected stock market transactions were generally much more complex than one would expect: with an average of 17 page references per transaction they were about 3 to 5 times heavier than the debit/credit transaction of the TPC–A benchmark [Gr91]. The standard deviation of the transaction length (in terms of page references) was 46, indicating a highly diverse workload. The stock market application comprised 51 different transaction types, one of which had an average length of 160 page references. There were nine occurrences of transactions that reference more than 500 pages, probably because of scans over the member records of large set occurrences. Figure 4 summarizes the access characteristics of the overall stock market workload and the top five transaction types in terms of the total number of page references per transaction type. The standard deviation of the transaction lengthof individual transaction types was relatively low for almost all transaction types, so that transaction length predictions based on average values promised to be meaningful.

Transaction type	Number of invocations	Total number of page references	Fraction of load (in terms of page references)	Number of page references min	mean	max.	standard deviation	Fraction of write accesses
all	2163	36619	100 %	1	16.9	681	46.1	21 %
BO330–331	51	8160	22 %	1	163.3	681	248.5	0 %
BO110–115	368	6984	19 %	1	19.0	72	10.7	34 %
BO110–113	354	5499	15 %	1	15.5	66	10.4	34 %
BO110–111	269	2447	7 %	1	9.0	54	6.4	0 %
BO210–212	178	2166	6 %	1	12.1	49	8.78	30 %
others	952	12402	31 %	1	12	278	21.2	22 %

Figure 4: Description of the Transaction Types in the Workload

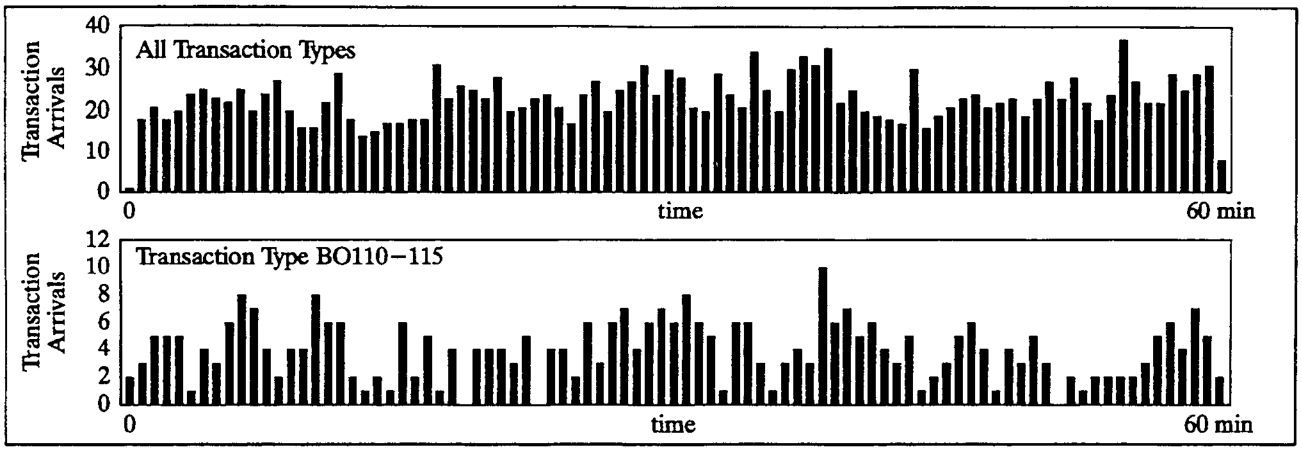

Figure 5: Temporal Distribution of Transaction Arrivals

The temporal distribution of the transaction arrivals is shown in Figure 5. The average interarrival time of stock market transactions was 1.2 seconds. The stochastic distribution of the interarrival time corresponds approximately to an exponential distribution function. However, the one-hour trace also exhibited some bursts of transaction arrivals, which cannot be modeled by a probabilistic distribution function. These bursts of arrivals were even more obvious for individual transaction types, as shown in Figure 5. Even more interestingly, there seemed to be a strong correlation in the load peaks of some transaction types, whereas the load peaks of other transaction types were uncorrelated or even seemed to alternate.

Such dynamic fluctuations in the transaction arrival rate and the transaction mix, which are not properly modeled by stochastic functions, were in fact the main motivation for using real-life traces to drive our performance evaluation. In the performance experiments that we performed, transaction arrivals were generated at the same relative points of time at which they occurred in the original trace, and their page references were fed into our testbed as fast as possible. We note in passing that the real OLTP system of the bank did not show any significant data contention problems during the one-hour trace period, since it used much faster hardware (Unisys mainframe) and was operating at a fairly restrictive DMP setting. The fact that our experiment was run on a parallel computer with relatively slow CPUs increased the degree of data contention in the workload. Note, however, that out experimental platform was able to process all transactions within one hour (i.e., the duration of the trace).

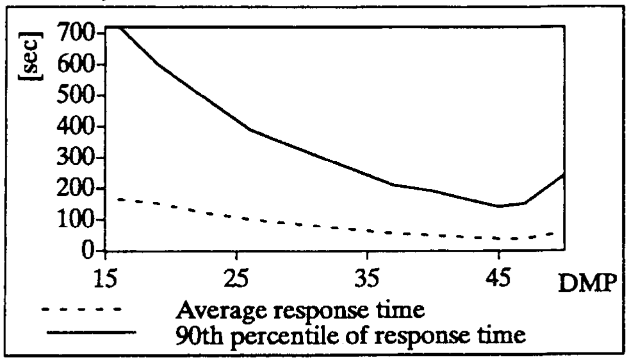

Figure 6: Transaction Response Time of the DMP Method

5 Performance Results

5.1 Reference Experiment

This subsection describes a refernce experiment that served as a yardstick against which our adadptive load control method was measured. In a series of runs, we employed the DMP load control at different DMP settings. Figure 6 shows the transaction response time as a function of the DMP values. The optimum DMP was 45, with an average response time of 37 seconds and a 90th percentile of 140 seconds.

There was a fair amount of data contention despite the fact that load control was in effect.[3] The number of lock waits (i. e., lock requests that led to transaction blocking) was 485, and the transactions spent a large fraction of their response time waiting for locks (about 36 percent on average). In addition, there was a fairly high number of deadlocks (79 deadlocks), which temporarily increased the CPU load due to the reprocessing ofaborted transactions. It may be worthwhile to mention that the observed high number of deadlocks is in contrast to the folk wisdom that deadlocks are infrequent [Gr81]. Recall, however, that the transactions in our experiment were much more complex than, for example, debit/credit transactions (see Section 4.2), and that many deadlocks may have been caused by specific reference patterns (see also [PR83] for similar observations on deadlock frequency in real-life workloads).The degree of data contention was reduced at lower DMP values; however, then the waiting time in the BOT queue became a significant factor so that the response time increased again. Figure 6 shows that the performance is quite sensitive to changes of the DMP; so it is not exactly easy to "guess" an appropriate DMP setting. Note that the experimentation by which we determined the optimal DMP is usually too expensive to be applied in a real-live production environment.

3. To verify that the system was indeed bound by data contention rather than resource contention (i.e. CPU or disk contention), we also ran an experiment in which all exclusive lock requests were replaced by shared lock requests. The avarage response time of this experiment was 5.52 seconds, and the 90th percentile was 7.9 seconds; there was no obvious bottleneck in this case.

	Optimal DMP method	Adaptive load control method	Admission control only
Avg. response time [sec]	37	24.5	46.0
90th percentile of response time [sec]	140	80	200
Number of non-admissions	470	6	36
Number of cancellations	---	18	---
Number of deadlocks	79	61	76
Avg. BOT queue wait time[sec] per transaction	1.34	0.02	0.22
Number of lock waits	485	448	446
Avg. lock wait time [sec] per transaction	13.6	11.0	15.6
Avg. DMP	16.0	15.1	14.8
Avg. conflict ratio	1.65	1.12	1.09

Figure 7: Performance of Different Load Control Methods

5.2 Performance of the Basic Load Control Method

In this subsection we compare the basic version of our adaptive load control method to the best possible DMP method. Our method used the admission and cancellation policy as described in Section 2. In addition, we included a version without cancellation control in the comparison. This version is supposed to be roughly similar to the feedback method of [HW91], which does not use cancellation control either. Figure 7 shows the average response time and the 90th percentile of the response time as well as further performance details for the three methods under comparison. The BOT queue wait time denotes the total time that a transaction spent waiting in
the BOT queue for one of the following three reasons: 1) the transaction is not admitted immediately upon its arrival, 2) the transaction is not admitted immediately for restart after having become a cancellation victim, or 3) the transaction is not admitted immediately for restart after having become a deadlock victim.[4] The main observations from this experiment are the following:

- The adaptive load control method clearly outperforms the (best possible) DMP method. It gains a factor of 1.5 in terms of the average response time and a factor of 1.8 in terms of the 90th percentile of the response time. The performance gains of our method are further illustrated by considering individual transaction types. Figure 8 shows the response time results for the four most dominant transaction types (i.e., the top four transaction types of Figure 4). For the very long read-only transaction type BO330-331, the performance of the DMP method was actually quite comparable to that of the adaptive load control method, both in terms of average response time and the 90th percentile. The performance penalty of the DMP method was much more significant for short transaction types (up to a factor of 4). The transaction type with the highest response time degradation was BO110-111, a frequent but fairly short read-only transaction, which one would usually consider harmless. Obviously, the transactions of this type suffered from performance problems that were caused by other transaction types (see below for further explanation). Such a situation is not only counterintuitive, it is also unacceptable in practice. In on-line transaction processing, it is especially important to guarantee good response time for the frequent "bread-and-butter" transactions, whereas for complex decision-support queries one would be willing to accept a higher variance of the response time. Note that the adaptive load control method behaved more intuitively in the sense that the response time of the various transaction types was roughly proportional to the complexity of the transactions.
- The reasons for the bad performance of the DMP method are twofold and have to do with the dynamic fluctuations of the workload (i.e., an inherent characteristic of most real-life workloads).
  1. At certain points in the course of the experiment, the rapid arrival of many potentially conflicting transactions produces a data-contention potential that requires very restrictive admission control; otherwise the system would possibly suffer DC-thrashing. However, since the DMP limit of the optimal DMP method is relatively high, the degree of data contention becomes quite critical, at least temporarily.
  2. At other points in the experiment, there are bursts of transaction arrivals which, however, may be quite uncritical in terms of their conflict potential (e.g., peaks of the read-only transaction type BO110-111). The response time of these transactions degrades because the transactions spend a long time waiting in the BOT queue, despite the fact that they are uncritical in terms of data contention. This effect is even amplified by the other type of load peaks, which really bear the risk of DC thrashing: the DC-critical load peaks lead to a backlog of non-admitted transactions, and the speed at which this backlog is processed is limited by the fixed DMP setting. The DMP method is not flexible enough to increase the DMP temporarily even if all currently executing transactions were read-only.

The above consideration shows a fundamental problem of the DMP method. If the DMP limit were set higher, then the DC-critical load peaks may indeed cause thrashing; if, on the other hand, the DMP limit were set lower, then the DC-uncritical load peaks would increase the unnecessary BOT queue wait time even fur-

4. Recall from Section 2.2.2 that deadlock victims may be delayed intentionally even if the conflict ratio has dropped below the critical threshold.

	Avg. length	Write fraction	Optimal DMP method		Adaptive load control method		Admission control only	
			Avg.	90thprecentile	Avg.	90thprecentile	Avg.	90thprecentile
BO330-331	163	0	102	277	64	231	78	309
BO110-115	19.0	0.34	53	245	33	87	52	296
BO110-113	15.5	0.34	46	161	18	53	38	114
BO110-111	9.0	0	44	153	10	41	30	69

Figure 8: Response Time [sec] of the Four Most Dominant Transaction Types

ther. So, whatever the "optimal" DMP is, it is a statically fixed limit and, therefore, merely a (bad) compromise for the different load situations that occur in the one-hour trace.

Our adaptive load control method, on the other hand, adapts the DMP to the current load dynamically and automatically. In the course of the experiment, the DMP varied between 1 and 50. The conflict ratio occasionally exceeded the critical threshold and reached a maximum of 10; however, the average conflict ratio of 1.127 was well below the critical threshold.

- The adaptive method with admission control only (i.e., without cancellation) performed about equally badly as the DMP method. Not having the option to cancel transactions in DC-critical situations seems to make the adaptive approach less reactive to dynamic fluctuations of the transaction mix.

5.3 Performance of the Method Exploiting Advance Knowledge

In this subsection we discuss the performance impact of expoiting predictions of the transaction lengths. We compare the basic version of the adaptive load control method (i.e., the "Ignorant" variant of Section 3) to the "Smart" and "Intelligent" methods described in Section 3. Recall that the Smart method is based on average lengths of transaction types, whereas the Intelligent method considers the average number of locks for each pair of transaction type and area (i.e., database partition). Orthogonally to these degrees of exploiting advance knowledge, we considered two options for the queueing discipline of the BOT queue: FIFO as in the basic version of our method or non-FIFO as described in Section 3. Figure 9 shows the response time results of the resulting five methods Ignorant (i.e., the basic version), Smart/FIFO, Smart/non-FIFO, Intelligent/FIFO, Intelligent/non-FIFO.

Some of the methods that exploit advance knowledge performed even better than the Ignorant method. The Intelligent method improved the average response time by about 20% and the 90th percentile of the response time by about 40%. The Smart method, on the other hand, performed worse than the Ignorant version. One of the pitfalls of attempting to be smart was the inaccuracy of the predictions of transaction lengths. Using average lengths of transaction types in the estimation of the conflict ratio seemed to overestimate the expected degree of data contention. Therefore, the Smart method held too many transactions in the BOT queue, and thus increased the transaction response time in periods that appeared to be DC-critical but were actually uncritical. The combination of the Smart method with a non-FIFO queueing discipline, on the other hand, admitted too many transactions in DC-critical periods. The reason for this was that, especially in DC-critical periods, the BOT queue contained enough transactions of almost any type. The non-FIFO method could then select for admission many transactions with a low predicted length. However, by the law of statistics, some of these admitted transactions were in fact longer than the average length of the corresponding transaction type. So, attempting to be smart and unfair (i.e., non-FIFO) did not pay off. Note, however, that both Smart methods were at least as good as the DMP method; so our method is quite robust even in the presence of inaccurate predictions.

The Intelligent method, on the other hand, seemed to estimate the expected changes of the conflict ratio quite accurately, based on the average number of locks for each pair af transaction type and database area. In this case, the non-FIFO version performed slightly better than the FIFO method. However, before drawing general conclusions on this issue, further performance studies are needed.

6 Conclusion

We presented an adaptive load control method for the avoidance of data-contention thrashing. This method has been stress-tested and its performance has been evaluated based on real-life page reference traces from an on-line transaction processing system. It was shown that our method can cope well with the diversity of the transaction mix and the dynamic fluctuations of the transaction arrivals. The adaptive load control clearly outperforms the best possible DMP method, since it can dynamically adapt the system to the temporal changes of the transaction mix. The performance of our method may be improved even further

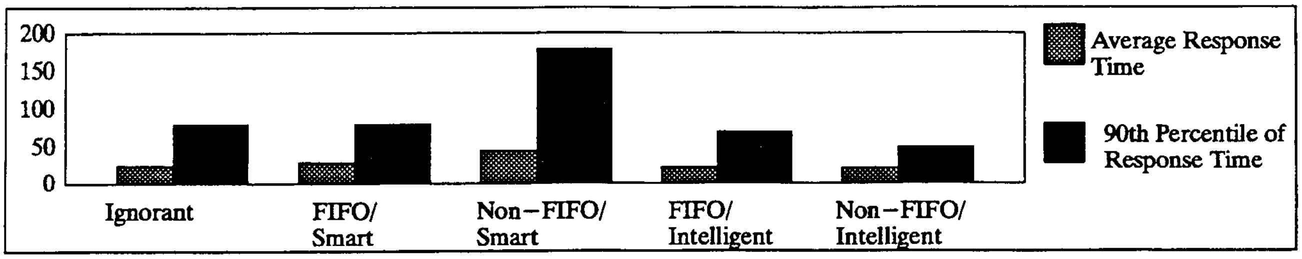

Figure 9: Response Time of the Methods with Exploitation of Advance Knowledge

by exploiting predictions of the lengths of transactions. However, our method does not depend on the availability of such advance knowledge; it is fairly robust without predictions or with inaccurate predictions.

More experimental work is needed to quantify the advantages and disadvantages of the various options of our load control method. The next step is to extend our performance evaluation by including the half–and–half method and possibly other proposals for data–contention load control. Among the issues that we further plan to investigate are the impact of alternative scheduling policies, such as prioritizing transactions that hold many locks, and the potential benefit of partial rollbacks [Mo92]. The idea of the latter is to undo a cancellation or deadlock victim only back to the point where its "most critical" locks can be released, that is, those locks that block other transactions. This option would potentially save work in the reprocessing of aborted transactions. Finally, we have begun to address the problem of load control for memory contention, based on DBMIN–like buffer management architectures [CD85, FNS91, YC91], and we plan to study the interaction of this load control component with the data–contention load control.

Our ultimate long–term goal that we pursue in the COMFORT project [Wei90] is to automate as many database performance tuning decisions as possible, thus simplifying the tricky job of system administrators and human tuning experts. The work presented in this paper is a step towards this ambitious goal.

7 Acknowledgements

We would like to thank the Union Bank of Switzerland (Schweizerische Bankgesellschaft) for supporting this work, and especially Dr. Stefan Gyr for providing us with the trace data and for his cooperation. We would also like to thank Christof Hasse for his help on troubleshooting various performance bugs of our prototype. Finally, we would like to thank Dr. Alex Thomasian for his comments on an earlier version of the paper.

References

[ACL87] Agrawal, R., Carey, M.J., Livny, M., Concurrency Control Performance Modeling: Alternatives and Implications, ACM TODS 12 (1987), 4

[ACM87] Agrawal, R., Carey, M.J., McVoy, L.W., The Performance of Alternative Strategies for Dealing with Deadlocks in Database Management Systems, IEEE Trans. on Softw Eng. 13 (1987), 12

[BHG87] Bernstein, P.A., Hadzilacos, V., Goodman, N., Concurrency Control and Recovery in Database Systems, Addison–Wesley, 1987

[BSR80] Bernstein, P.A., Shipman, D.W., Rothnie, J.B., Concurrency Control in a System for Distributed Databases (SDD–1), ACM TODS 5, (1980), 1

[Bu89] Buchmann, et al., Time–Critical Database Scheduling: A Framework for Integrating Real–Time Scheduling and Concurrency Control, IEEE Data Engineering Conf., 1989

[CD85] Chou, H.–T., DeWitt, D.J., An Evaluation of Buffer Management Strategies for Relational Database Systems, VLDB Conf., 1985

[CKL90] Carey, M.J., Krishnamurthi, S., Livny, M., Load Control for Locking: The 'Half–and–Half' Approach, ACM PODS Conf., 1990

[De68] Denning, P., Thrashing: Its Causes and Prevention, AFIPS Conf.., Vol. 33, 1968

[FNS91] Faloutsos, C., Ng, R., Sellis, T., Predictive Load Control for Flexible Buffer Allocation, VLDB Conf., 1991

[FR85] Franaszek, P., Robinson, J., Limitations on Concurrency in Transaction Processing, ACM TODS 10 (1985), 1

[FRT91] Franaszek, P., Robinson, J., Thomasian, A., Wait Depth Limited Concurrency Control, IEEE Data Eng. Conf., 1991

[Gr81] Gray, J., et al., A Straw Man Analysis of Probability of Waiting and Deadlock, IBM Res. Rep. RJ3066, San Jose, 1981

[Gr91] Gray, J. (Ed.), The Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufmann, 1991

[Hae87] Haerder, T., On Selected Performance Issues of Database Systems, Invited Paper, 4th GI/ITG Conf. on Performance Modeling of Computing Systems, Springer, 1987

[Hei91] Heiss, H.–U., Overload Effects and Their Prevention, Performance Evaluation 12 (1991)

[Hi89] Highleyman, W.H., Performance Analysis of Transaction Processing Systems, Prentice Hall, 1989

[HW91] Heiss, H.–U., Wagner, R., Adaptive Load Control in Transaction Processing Systems, VLDB Conf., 1991

[JTK89] Jenq, B.P., Twichell, B., Keller, T., Locking Performance in a Shared Nothing Parallel Database Machine, IEEE Data Eng. Conf., 1989

[Mo92] Mohan, C., et al., ARIES: A Transaction Recovery Method Supporting Fine–Granularity Locking and Partial Rollbacks Using Write–Ahead Logging, ACM TODS 17,1 (1992)

[MW91] Moenkeberg, A., Weikum, G., Conflict–driven Load Control for the Avoidance of Data–Contention Thrashing, IEEE Data Eng. Conf., 1991

[PR83] Peinl, P., Reuter, A., Empirical Comparison of Database Concurrency Control Schemes, VLDB Conf., 1983

[Ra89] Rahm, E., A Framework for Workload Allocation in Distributed Transaction Processing Systems, to appear in: Journal of Systems and Software

[Reu86] Reuter, A., Load Control and Load Balancing in a Shared Database Management System, IEEE Data Eng. Conf., 1986

[Sha92] Shasha, D., Database Tuning: A Principled Approach, Prentice Hall, 1992

[TGS85] Tay, Y., Goodman, N., Suri, R., Locking Performance in Centralized Databases, ACM TODS 10 (1985), 4

[TR90] Thomasian, A., Ryu, I.K., Performance Analysis of Dynamic Locking with the No–Waiting Policy, IEEE Trans. on Softw. Eng. 16, (1990), 7

[TR91] Thomasian, A., Ryu, I.K., Performance Analysis of Two–Phase Locking, IEEE Trans. on Softw. Eng.. 17 (1991), 5

[Th91] Thomasian, A., Performance Limits of Two–Phase Locking, IEEE Data Eng. Conf., 1991

[Th92] Thomasian, A., Thrashing in Two–Phase Locking Revisited, IEEE Data Eng. Conf., 1992

[YBL88] Yu, P.S., Balsamo, S., Lee, Y., Dynamic Transaction Routing in Distributed Database Systems, IEEE Trans. on Softw. Eng. 14 (1988), 9

[YC91] Yu, P.S., Cornell, D.W., Optimal Buffer Allocation in a Multi–Query Environment, IEEE Data Eng. Conf., 1991

[Wei90] Weikum, G., et al., The COMFORT Project: A Comfortable Way to Better Performance, Tech. Rep., ETH Zurich, 1990