

The Use of Contention-Based Scheduling For Improving The Throughput of Locking Systems

Samuel Kaspi

Victoria University, Business faculty, Department Information Systems
Australia
email: samuel.kaspi@vu.edu.au

Abstract. The behavior of locking systems suggests that controlling concurrency by transaction type rather than by the aggregate number of transactions could increase throughput significantly. Accordingly, we propose a system for manipulating concurrency by transaction type. This system comes in three parts. Firstly a mechanism that could be used to determine transactions' contention and to sort them into appropriate queues before they are allowed into the system. Secondly an algorithm for determining the maximum number of each transaction type allowed into the system and thirdly, a set of priorities to determine which transaction would be successful in the event of a clash over an object. We present empirical results which show that our proposed system does indeed dramatically increase aggregate throughput.

Keywords: throughput, contention, transaction type, transaction priority, contention-based scheduling, aggregate contention.

1 Introduction

Increased hardware capacity and new applications such as e-commerce have brought about a demand for higher transaction throughputs. Because of the commercial importance of disk-based locking systems, algorithms that improve the performance of such systems are required. In this paper, we investigate how the throughput of such systems can be improved by manipulating contention. The advantages of the algorithms proposed in this chapter is that they dramatically increase the peak throughput of locking systems and that their implementation requires very little modification of the standard lock based DBMS' currently in operation.

Our paper is organized as follows. In section 2, we analyze the behavior of locking systems and present equations for measuring contention and predicting behavior by transaction type. In this section we also present proposals to significantly improve performance of locking systems by manipulating concurrency by transaction type. In section 3, we describe our test systems and in section 4 we verify the performance gains that can be achieved by our manipulation of the locking system.

2 Improving the Performance of locking by Manipulating Contention

Under standard locking concurrency control, arriving transactions are allowed into the system on a first come first serve basis. Only the permitted level of concurrency restricts entry into the system. We propose a contention-based scheduler that measures transactions' contention as they arrive sorts these transactions into queues of transactions of similar contention and then manipulates the number of transactions allowed into the system by the contention class. By doing so, the contention-based scheduler can dramatically increase the system's throughput. The rationale behind this manipulation is that lower contention transactions have a lower probability of being involved in a conflict. It is thus possible to increase the effective level of concurrency by permitting a high proportion of low contention transactions entry into the system while restricting the number of high contention transactions allowed entry into the system. This is because the increase in throughput gained by increasing the number of lower contention transactions is larger than the throughput lost by reducing the higher contention transactions.

A description of the operation of the contention-based scheduler is as follows. As transactions arrive, the scheduler does a pre-fetch for each transaction to establish its contention and then places it on its appropriate queue. Such a pre-fetch is only for the purposes of estimating contention and does not actually acquire or release locks. Having sorted transactions according to their contention, the scheduler ensures that no more than the prescribed number of each transaction type is allowed into the system. In order to be viable, this pre-fetch needs to be effected at a minimal cost. Implementing the pre-fetch as a memory only operation can achieve this. Here, a transaction does a pseudo-execution. If a required item is found in cache its contribution to contention is calculated. If an item is not found in cache, its contribution to contention is assigned a default value based on the average contention of objects found on disk. Where both the number and contention of objects that need to be acquired from disk is unknown because the choice of object is dependent on predicate satisfaction, the estimation of both the number and contention of the objects can be assigned a default value.

One heuristic for determining this default could be that where the number of objects required from disk is unknown, this number could be set by dividing the number of objects a transaction has acquired from cache by the proportional hit rate of objects found in cache. Since in general, all high contention items are in cache, any imprecision in assigning a default contention to items not found in cache is likely to be of minimal consequence. The robustness of the contention-based scheduler to imprecision will be shown later in this chapter when the results for imprecise schedulers are presented.

2.1 The Measurement of Contention

In order to determine how many transactions of each type to allow into the system, the contention based-scheduler needs to be able to measure contention. While there are several equations that measure contention such as those found in [1], [3] and [4] all of these have deficiencies Vis a Vis our requirements. In [3] and [4] the measurement of contention is dependent on the level of concurrency while for our requirements we need a measure of contention that is independent of concurrency. Our equation for measuring any transaction class t_a 's contention is –

$$p(t_a) = S \sum_{k=1}^j \text{freqAccess}(O_k) \quad (1)$$

Here O_i to O_k are the objects that transaction t_a requires, *freqAccess* is each object's probability of being required in any access and S is the mean size of all transactions in the system. In a database where all objects are accessed with the same probability, any object's *freqAccess* is simply $1/D$ where D is the size of the database. In a database where objects do not have the same probability of being required in an access, each object's *freqAccess* is determined historically. Alternatively, a database can be partitioned into groups of objects. Here, where D_x is the number of elements in the group that O_i belongs to and k is the probability that in any access the object required will come from D_x , an object O_i 's *freqAccess* can be calculated by -

$$\text{freqAccess}(O_i) = k/D_x \quad (2)$$

The above assumes that each object in a group has the same probability of being required in an access as any other object in that same group and that each object in a group has a different probability of being required in an access to any object in any other group. Our estimation for the first cycle throughput of any transaction type, say transaction class t_a , is -

$$n r(t_a) (1 - p(t_a)/2)^{(n-1)} \quad (3)$$

This modifies the equation in [1] by adding the expression $r(t_a)$ and replacing mean contention p , with $p(t_a)$. Here, $r(t_a)$ is the proportion of transactions that belong to class t_a , while $p(t_a)$, as per equation 1, is the contention of each transaction belonging to transaction class t_a . As in [1], n is the level of concurrency. These changes allow for measurement of throughput by transaction type rather than in aggregate.

2.2 Determining the Maximum Number of Each Transaction Type Allowed into the System

The algorithm presented in this section is heuristic and while it improves performance it is not necessarily optimal. Our algorithm is as follows. Firstly we determine each transaction class' contention as per equation 1 and the mean system contention. We then determine the concurrency at an aggregate contention

of 2.6. The reason for choosing 2.6 as the aggregate contention is that it seems to give the best results. We then multiply the number of transactions derived above by the proportion of each transaction type giving $n(t_i)$ for any transaction type t_i . In our next step we calculate each transaction class' contention as if that class was the only transaction type in the system. Our approximation for this for any transaction type t_i is –

$$pIsolated(t_i) = p(t_i) (p(t_i) / pmean) \quad (4)$$

Next, we calculate each transaction class' $pRatio$ by dividing its $pIsolated$ by the $pIsolated$ of the lowest contention class. We then derive each transaction class' $pContribution$. For any transaction class t_i ,

$$pContribution(t_i) = pRatio(t_i) n(t_i) \quad (5)$$

In our final step, for any transaction class t_i , where $r(t_i)$ is the proportion of transactions that are t_i , the maximum number of t_i transactions allowed into the system is –

$$max(t_i) = totContribution r(t_i) / pRatio(t_i) \quad (6)$$

2.3 Setting Transactions priority

While the algorithms presented above may increase aggregate throughput, they are unfair in that higher contention transactions "subsidize" lower contention transactions. To compensate the higher contention transactions we institute a priority system such that a transaction with a higher contention has a higher priority than a transaction with a lower contention. Thus, if a t_4 transaction conflicts with a t_3 transaction, it is the t_4 transaction which obtains the lock. If the conflict occurs after the t_3 transaction has gained its lock, then the t_3 transaction is rolled back to the point where it acquired the lock on the disputed item and the t_4 transaction is given the lock.

The effect of this prioritization is to increase the success rate of higher contention transactions without dramatically affecting the success rate of lower contention transactions. Consequently, the throughput of each transaction class under the scheduler should either increase or be close to the throughput for that transaction class under systems without the scheduler. A major advantage of this system, is that it requires very little modification to existing locking systems since virtually all existing DBMS' implementing locking concurrency control also implement rollback and priority. The only difference between rollback and priority in conventional systems and that suggested in this paper, is that with the contention-based scheduler a transaction's priority would be determined by its contention and rollback requires automatic checkpointing by the DBMS such that a checkpoint is automatically inserted into the transaction by the DBMS every time a new object is acquired. As well, to avoid problems associated with restoring rolled back values, all a transaction's updates are done

in its own workspace and are not applied to the actual data until committal. Thus, a rollback by a transaction only involves a move back to the checkpoint and does not require restoring original object values.

3 Test Systems

To determine the efficacy of our contention-based scheduler, we compare its performance when used in conjunction with locking to standard two-phase locking (2PL) without the scheduler and to optimistic and wait depth limited (WDL) concurrency control methods. Comparisons are made under a wide range of hardware configurations. All systems are tested for one second at each concurrency level. At each concurrency level for each system the results shown are an average over 40 runs.

3.1 Database and Transaction Models

For each of our tests there are two object stores available to transactions from which to choose their objects - D_1 , which contains 1000 objects, and D_2 , which contains a million objects. Two characteristics of objects in these data stores are firstly that objects contained in data stores are disjoint and secondly, given a data store D ; any two objects O_i and O_j in that data store have an equal probability of being required in any access.

Each of our systems contains four transaction types - t_1 , t_2 , t_3 and t_4 . These transaction types require 1, 2, 4 and 8 objects respectively from D_1 and 3, 6, 12 and 24 objects respectively from D_2 . t_1 transactions represent 20% of transactions, t_2 transactions represent 20% of transactions, t_3 transactions represent 35% of transactions and t_4 transactions represent 25% of transactions. Each transaction has three stages - initialization, processing and completion. The initialization stage, whether for new or restarted transactions requires 100000 instructions per transaction regardless of the transaction's size. This stage requires CPU processing only. Similarly, the completion phase requires 5000 CPU instructions per **committing** transaction. It is assumed that once a committed transaction is placed in a buffer, the system is ready for the next transaction; thus, no disk access is required in this phase. The processing phase requires 20000 CPU instructions per data item - this includes the overheads for concurrency control. These processing requirements are the same as found in [2] and [6]. Since we are interested in systems with high throughputs, a large number of transactions need to be available, thus the arrival rate of transactions is 18000 transactions per second arriving at a constant rate.

3.2 Hardware Subsystems

The hardware subsystems used for our tests are, a subsystem containing 96 processors each operating at 100 MIPS, a subsystem containing 96 processors

each operating at 200 MIPS, a subsystem containing 20 processors each operating at 1000 MIPS and a subsystem containing 10 processors each operating at 2000 MIPS. For each of these subsystems, the cache holds sufficient data to ensure that 0.625 of items required in a first accesses are in memory. For restarted transactions or transactions in process, all items required remain in cache and are not flushed till the transaction commits. These parameters are in accord with those found in [2] and [6].

Each hardware subsystem has 15880 disk arms except the 96 processor 100 MIPS per processor subsystem which has 7940 disk arms. The determination of this number of disk arms is based on the algorithm used in [2] and [6] which applies Little's law using total CPU processing power, the number of expected disk accesses and a CPU and disk utilization ratio of 75/20 as parameters. All disk accesses are uniformly distributed (no skew). Each disk access requires 15 milliseconds.

3.3 Concurrency Control

As indicated above, the performance of our contention-based scheduler when used in conjunction with a locking system is compared against the performance of standard 2PL, WDL and optimistic concurrency control mechanisms. The 2PL mechanism includes a deadlock breaking mechanism and is standard except that all requests for locks are requests for a write and that the granularity of lock acquisition and release is a single lock.

When used with the contention-based scheduler, the locking mechanism allows for rollback in order to accommodate the contention-based scheduler's priority system. While rollback violates one of the principles of 2PL – that is, that all locking is done in one phase and unlocking in another, it is so widely used, that we do not consider this violation to be a serious problem to our algorithms. The maximum number of transactions of each type allowed into the system by our algorithm is 734, 184, 80 and 14 for transaction types t_1, t_2, t_3 and t_4 respectively. As well, the contention-based scheduler is tested for each of two assumptions – that the contention-based scheduler measures contention perfectly and alternatively, that the contention-based scheduler measures contention imperfectly.

To simulate error, 20% of t_1 transactions are placed on the t_2 transaction queue. 10% of t_2 transactions are placed on the t_1 queue and 10% of t_2 transactions are placed on the t_3 queue. 10% of t_3 transactions are placed on the t_2 queue and 10% of t_3 transactions are placed on the t_4 queue. 20% of t_4 transactions are placed on the t_3 transaction queue. When a transaction completes, its actual contention is re-measured and the appropriate number of transactions of each type in the system is re-calculated. While the selection of 20% as the margin of error for testing is arbitrary, we believe that this margin is as high as could reasonably be expected to occur.

The WDL concurrency control mechanism is as described in [2]. As with the locking system, all requests for locks are requests for a write and the granularity of lock acquisition and release is a single lock. Access invariance and the pre-fetch properties consequent to it are assumed to hold. Thus, for these systems, restarted transactions find all the objects that they acquired prior to restarting in cache. Since access invariance only holds at low concurrencies, the maximum concurrency at which these systems are tested is 100.

The optimistic method we use is a hybrid die-kill system as outlined in [2] and [6]. In this system, transactions in their first phase die at the end of their processing cycle if they have conflicted with a previously committed transaction. By dying after processing, transactions are able to pre-fetch all their required objects. Transactions that have died and are restarted are killed if they conflict with a previously committed transaction. This variation allows transactions to maximize the benefits of pre-fetching while minimizing the cost of wasted work for transactions that have already pre-fetched all their data. This of course assumes that access invariance and its consequent pre-fetch property holds. Again, as for WDL, to facilitate access invariance, the maximum concurrency is 100.

4 Performance Results

The results presented in this section, compare the results of locking systems using our scheduling algorithms against the best results achieved by the 2PL, WDL and optimistic systems described in the previous section. The peak results for the standard 2PL method is achieved at a concurrency of 70. The peak results for WDL and optimistic concurrency are achieved at the maximum concurrency at which the systems are tested that is, 100. Because the contention-based scheduler changes both the proportion and priority of transactions in the system, it is likely that it would cause distortion in the throughput of each transaction type. Consequently, it is possible that it could increase total throughput while at the same time reducing the throughput of some transaction types. We therefore compare the performance of our systems both in total and by transaction type. Thus, figure 1 shows total throughput while figure 2 breaks up total throughput by transaction type.

As figure 1 shows, the improvement in total throughput achieved by our scheduling algorithms Vis a Vis standard 2PL is quite large under all hardware configurations with the improvement ranging from around 800% to around 1500% . The breakdown of total throughput in figure 2 shows that as expected, this improvement is most pronounced for the lowest contention t_1 transactions, substantial for the next lowest contention t_2 transactions and significant for the medium contention t_3 transactions. The only transaction type where standard locking outperforms our scheduler is the highest contention t_4 transactions and here, the superiority of the standard locking system is marginal.

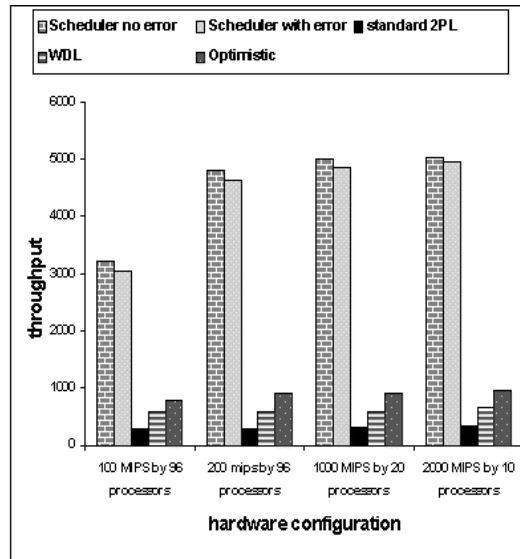


Fig. 1. A comparison of total throughputs

The results also indicate that the performance of our scheduling and rollback algorithms is very robust even when one allows a significant margin of error in the classification of transactions' contention. Thus, under all the tested hardware configurations, total throughput when error in measurement occurs is very similar to total throughput under equivalent hardware configuration when no error in measurement occurs. The breakdown of total throughput by transaction type in figure 2 indicates that this robustness extends to all transaction types in all the hardware configurations tested.

While our contention-based scheduler algorithms perform best when compared to the results achieved by standard 2PL concurrency management, their performance Vis a Vis WDL and optimistic concurrency control is also excellent achieving a higher total throughput than is achieved under either WDL or optimistic die-kill. As in the comparison with the 2PL systems, the performance advantage of our scheduler Vis a Vis WDL and optimistic die-kill is concentrated in the throughputs of the lower contention t_1 and t_2 transactions.

An interesting result is that in the throughput of the highest contention t_4 transactions, to the concurrencies tested, the optimistic die-kill systems outperform WDL. The explanation for this is that in the die-kill systems, restarted transactions have effectively pre-fetched all their data and can re-execute entirely in memory. Since the time taken to re-execute in memory for the longer transaction is less than the time that shorter transactions executing for the first time require when disk access is taken into account, longer transactions though

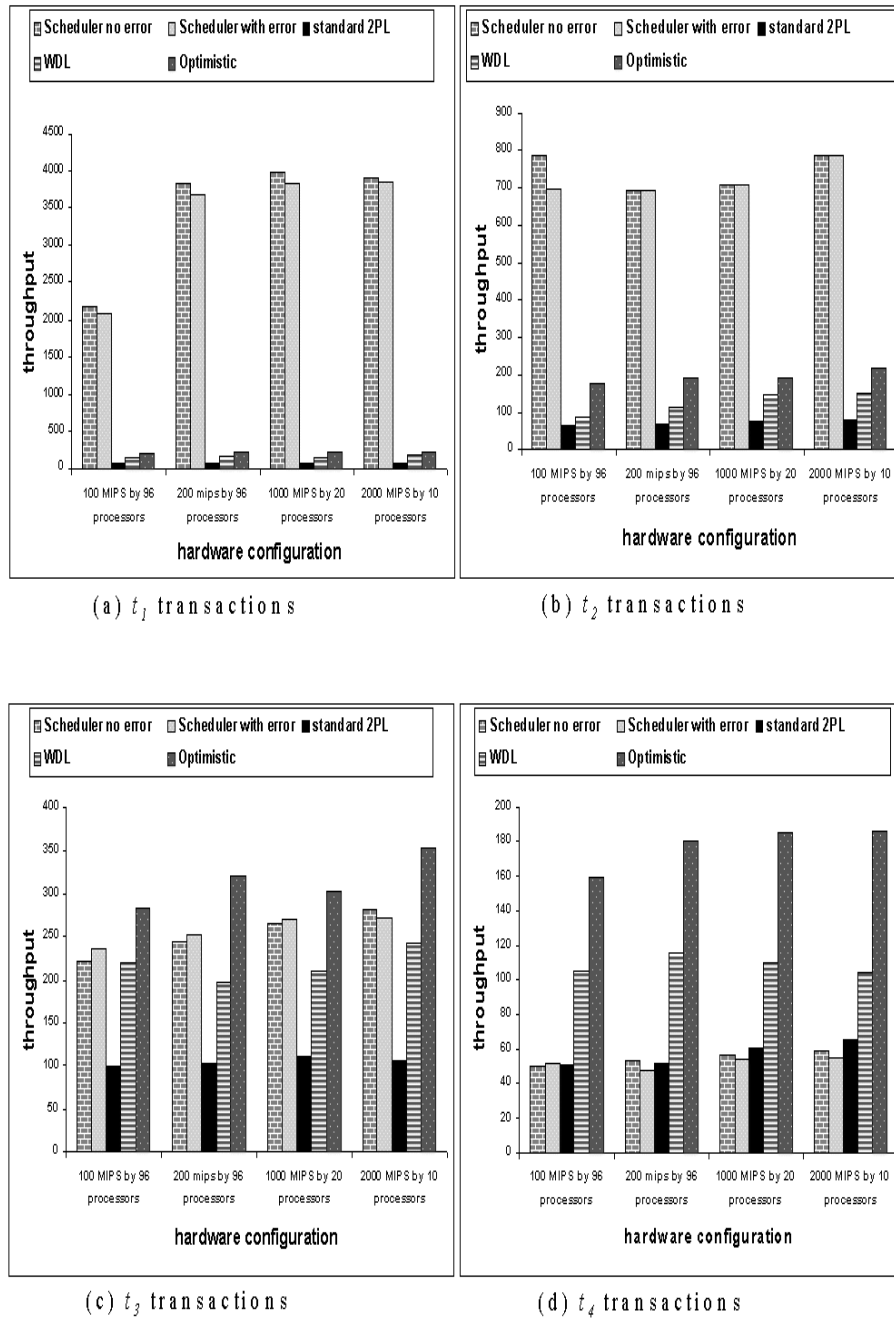


Fig. 2. Breakdown of total throughput by transaction type

more likely to die initially are more likely to succeed on restart. This advantage in execution time on restart along with the fact that optimistic systems are essential blocking systems, seems to be greater than the priority advantage conferred on the larger transactions by WDL.

5 Conclusions

In this paper we introduced the contention-based scheduler. This scheduler operated by measuring transactions' contention as they arrived, sorted transactions into queues of similar contention and then manipulated the number of transactions allowed into the system by contention class. For its operation, the contention-based scheduler needed a mechanism that could enable it to estimate transactions' contention reasonably cheaply, a method for calculating transactions' contention that was independent of concurrency, a method for determining the maximum number of transactions of each type allowed into the system and a method to compensate the higher contention transactions for their subsidy to lower contention transactions. All these requirements were addressed in section 2.

Tests presented in this paper showed that the contention-based scheduler substantially outperformed standard 2PL concurrency control in a wide variety of disk-based hardware configurations and that its performance was competitive when compared to WDL and optimistic concurrency control. Further, The results presented in indicated that the performance of our scheduling algorithms is very robust even when one allows a significant margin of error in the classification of transactions' contention.

References

1. Franaszek, P., and Robinson, J.T.,: Limitations of Concurrency in Transaction Processing. In: *ACM TODS*, Vol. 10 , No.1, March 1985, 1 - 28
2. Franaszek, P., Robinson, J.T. and Thomasian, A.,: Concurrency Control for High Contention Environments. In: *ACM TODS*, Vol.17, No.2, June 1992, 304 - 345
3. Thomasian, A., "Performance Limits of Two Phase Locking ", 7th IEEE International Conference on Data Engineering, Kobe, Japan, 1991.
4. Thomasian, A., and Ryu, K.,: Performance Analysis of Two-phase Locking. In: *IEEE Transactions on Software Engineering*, Vol. 17 No. 5, , May , 1991, 386-402
5. Thomasian, A.,: A performance Comparison of Locking Methods with Limited Wait Depth. In: *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9 No. 3, May/June , 1997, 421-434
6. Thomasian, A.,: Distributed Optimistic Concurrency Control Methods for High-Performance Transaction Processing . In: *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10 No. 1, January/February , 1998, 173-189