Analytical Modeling of Lock-based Concurrency Control with Arbitrary Transaction Data Access Patterns

Pierangelo Di Sanzo, Roberto Palmieri,
Bruno Ciciani, Francesco Quaglia
Sapienza, Università di Roma, Italy

Paolo Romano
INESC-ID, Lisbon, Portugal

ABSTRACT

Nowadays the 2-Phase-Locking (2PL) concurrency control algorithm still plays a core rule in the construction of transactional systems (e.g. database systems and transactional memories). Hence, any technique allowing accurate analysis and prediction of the performance of 2PL based systems can be of wide interest and applicability. In this article we present an accurate analytical model of 2PL concurrency control, which overcomes several limitations of preexisting analytical results. In particular our model captures relevant features of realistic data access patterns, by taking into account access distributions that depend on transactions' execution phases. Also, our model provides significantly more accurate performance predictions in heavy contention scenarios, where the number of transactions enqueued due to conflicting lock requests is expected to be non-minimal. The accuracy of our model has been verified against simulation results based on both synthetic data access patterns and patterns derived from the TPC-C benchmark.

Categories and Subject Descriptors

D.4.8 [**Performance**]: Modeling and prediction

General Terms

Algorithms

1. INTRODUCTION

Transactional systems, such as DBMSs, play the role of core software components for the design and implementation of a wide spectrum of modern applications (e.g. Web-based e-Business applications). Hence, any support for capacity planning, tuning and configuration of these systems is a fundamental issue to address.

To provide such supports, a highly critical aspect to cope with is to capture the complex effects deriving from the selected concurrency control mechanism, which is used to regulate concurrent accesses to the data items according to the requested transactions' isolation level. Dealing with this issue, the traditional approach for serializable concurrency control is 2-Phase-Locking (2PL) [9], which is also widely adopted by a plethora of modern commercial products (such as IBM DB2, Informix, Sybase ASE and Microsoft SQL Server). For this concurrency control protocol we present in this article an innovative analytical model, which is able to overcome several limitations characterizing any preexisting analytical result we are aware of (e.g. [6, 13, 15, 23, 26]), thus providing a more accurate tool for the performance analysis of 2PL based systems.

Our model is characterized by two main innovative features. (A) Unlike previous analytical approaches, our model does not assume that data accesses within a transaction are identically distributed, instead it accounts for data access patterns that may vary depending on the current transaction execution phase. This allows the model to capture realistic patterns, since, as also specified by several benchmarks (e.g. TPC-C [24]), transactions typically follow a deterministic order in the access to the data items. For instance, in warehouse applications, an order may be forwarded to the warehouse (by updating a given tuple in the Orders table) only after having checked the address of the customer (by reading, in sequence, from the Customers and Addresses tables). We take such phenomena into account by modeling the whole set of data manipulations executed by a transaction as a sequence of distinct phases, each one possibly characterized by different data access pattern distributions. (B) The model does not assume that the maximum length of the queue of transactions waiting for a read/write lock is bounded by one unit. In other words, we can accurately capture heavy contention effects and their impact on the final perceived performance.

Clearly, the history of data accesses, and in particular the order according to which data items are accessed within different transaction phases, has a significant impact on the distribution of locks' duration and has consequently a strong impact on the transactions' conflict probability. Consider, for instance, two sets of tuples, say X and Y, which are always the first, respectively the last, ones to be accessed within a transaction. Being the duration of the locks maintained on the tuples of set X much longer (relatively speaking) than the duration of the locks on the tuples of set Y, it follows that the probability of contention on the tuples of set X will be much higher than the probability of contention on the tuples belonging to set Y. Unfortunately, the inability of existing 2PL models to capture such effects makes them subject to potentially remarkable errors. This is confirmed by

WOSP/SIPEW'10, January 28–30, 2010, San Jose, California, USA.

our experimental study (see Section 4), where we show how performance models of 2PL that do not account for phase dependent access distributions, exhibit (under literature typical system settings) up to 60% error on the evaluation of the system saturation point while varying the transactions' arrival rate. This leads these models to exhibit unacceptable errors on the evaluation of the system response time vs variations of the transactions' workload. As noted in point (A) above, this paper fills such a gap by introducing a novel modeling approach that captures scenarios with different data items' access probabilities in distinct transaction execution phases.

We have evaluated the accuracy of our analytical model via an extended simulation study relying on both: (i) Synthetic workload descriptions (e.g. in terms of machine instructions for specific operations within a transaction), some of which analogous to those used for the validation of previous analytical results coping with both 2PL and Optimistic concurrency control [26], and (ii) A workload we derived by abstracting the main features of the transaction profiles specified by the TPC-C benchmark [24].

By the results, we also show that, even in case of phase-independent transaction patterns (where the transaction locality over the data items does not depend on the specific execution phase), pre-existing analytical performance models (see, e.g., [26, 23]) can incur some divergences from simulated performance values. This occurs especially under heavy workloads. Our model does not exhibit such a divergence just thanks to its feature in point (B), which allows it to achieve a much higher accuracy in the modeling of scenarios characterized by non-minimal data contention, even in presence of phase-independent data access patterns.

The remainder of this paper is structured as follows. In Section 2 we discuss literature works on transactional systems' modeling/evaluation. The analytical performance model of 2PL is provided in Section 3. Finally, simulation results that validate the model are presented in Section 4. Section 5 concludes the paper.

2. RELATED WORK

Several literature results exist, which cope with the evaluation of transactional systems and concurrency control protocols.

Analytical modeling approaches have been presented in [15, 22, 23] for the case of centralized database systems, and in [6, 7] for the case of distributed/replicated databases. Also, in [26] a general methodology for modeling and analytical evaluation of centralized transactional systems is provided. Some of the previous results differ from our proposal in that they cope with modeling of optimistic concurrency control, while our interest is in 2PL. Similar considerations apply to the work in [13], where a performance model for snapshot-isolation Multi-Version concurrency control is provided, which is intrinsically different from 2PL and provides a different, non-serializable transaction isolation level [8]. On the other hand, compared to literature analytical results related to lock-based concurrency control (see, e.g., [20, 21, 23, 26]), our proposal exhibits the innovative features of capturing the effects of data access locality vs specific transaction execution phases, and removes the assumption of lock access queue bounded by one unit.

Concurrency control protocols, and their impact on performance, have been extensively studied also via simulation [3, 4, 11, 18, 19], which is a technique orthogonal to the analytical approach provided in this paper.

Finally, our work is also related to those in [2, 25]. However, differently from our proposal, those analytical results are not focused on complete performance models of concurrency control, since they are exclusively focused on the evaluation of storage management tradeoffs in multi-versioning schemes vs the data item update frequency.

3. THE ANALYTICAL MODEL

3.1 Main Assumptions and Considerations

In accordance with typical assumptions in previous analytical studies (e.g. [26]), we assume a hardware system where the CPU is modeled as an M/M/k queue, where k is the number of CPU-Cores, each of which has processing speed denoted as $MIPS$ (measured in terms of instructions per second), and where the disk has a fixed I/O delay denoted as $t_{I/O}$. Anyway, we underline that our focus is on the effects of data accesses and contention on logical resources, not on physical resources. In fact, the contribution we provide is orthogonal to the assumed model for the underlying physical system, given that our model for logical resources' contention can be actually coupled with different models for physical resources.

The transactional system handles a set of I items, each of which represents a unit of data that can be accessed by an operation within a transaction (e.g a tuple or a set of tuples in a table of a database, or an item in a software transactional memory). Also, transactions are processed according to the Strong Strict Two-Phase Locking (SS2PL) concurrency control protocol [9]. Unlike in 2PL, where locks can be progressively released during the so called shrinking phase of the transaction execution (typically corresponding to the bottom half of the transaction life), SS2PL maintains any lock till the end of the transaction. We decided to target SS2PL as it is the most diffused variant among (commercial) database systems relying on lock-based concurrency control, though our model could be naturally adapted to forecast the performance of other 2PL variants.

With locking protocols, a transaction can be aborted by the deadlock manager, however we ignore deadlock related aborts in our model since, as shown by previous studies [5, 14], their effects on the final perceived performance are typically negligible, unless the system is already close to saturation (in which case the saturating performance curve is already captured by our model) ([1]).

Further, we assume that the system is stable and ergodic, so that quantities like the contention probability and the mean transaction response time exist and are finite, and defined to be either long-run averages or steady-state quantities.

Finally, our analysis is based on an open model. This choice is motivated by the fact that open models are more suited for scenarios with a large number of users (like in, e.g., transactional applications over the Internet).

[1] We do not explicitly report deadlock statistics in our simulation results supporting model validation. Anyway, the frequency of deadlock occurrence is actually negligible also for the case studies we have considered.

3.2 Basic Analytical Model

In this section, we present a basic version of the analytical model relying on a workload characterized by a single transaction profile (a single transaction class). This is specified in terms of number of data item to be accessed and locality of the accesses in the different phases of the execution. Also, we assume that transactions arrive according to a Poisson process with mean value λ. The model extension coping with workloads entailing differentiated transaction profiles (multiple transaction classes) will be discussed in Section 3.3.

3.2.1 Transaction Execution Model

Each transaction consists of an initial begin phase, which is followed by a number of M execution phases, each one accessing in read or write mode a single data item, and finally by a commit phase. To execute a read operation, a transaction has to obtain a shared read-lock on the target data item. On the other hand, for write operations, exclusive write-locks are needed. Hence, each operation might entail a wait (block) phase in case the requested lock is currently unavailable.

During begin and commit phases, an exponentially distributed number of CPU instructions are executed, with mean values $nIST_b$ and $nIST_c$, respectively. Also, the execution of an operation is assumed to require an exponentially distributed number of CPU instructions with $nIST_o$ as the mean value. (The exponential assumption has been introduced to match the M/M/k model for the underlying CPU.) In case the access to a data item causes a buffer miss, a time $t_{I/O}$ is needed to fetch the data from disk. We denote the expected buffer hit probability as P_{BH}. Actually, we do not explicitly model the buffering policy and the related effects since several models have already been proposed to cope with the evaluation of hit probability vs the item popularity, see, e.g. [12], which is orthogonal to our study. Hence, P_{BH} will be considered as an independent parameter in our study. Finally, for simplicity, we do not explicitly model the I/O delay associated with the commit phase (e.g. the transaction log write onto stable storage). Anyway, given our disk model, this delay would only entail an additional latency term in the expression of the transaction execution time. Also, our assumption well fits scenarios where the writing of the transaction commit log is executed asynchronously, thus not directly contributing to the transaction latency.

On the basis of the previous considerations, a transaction can be modeled through a direct graph (see Figure 1), where the nodes represent different states of the transaction execution and the arcs represent state transitions. A label on an arc from a node p to a node q represents the transition probability from state p to state q. If the label is omitted, then the transition probability is intended to be 1. States labelled with *begin* and *commit* represent begin and commit phases respectively, while the state labelled with $\hat{k}$ represents the execution of the k^{th} operation, and, finally, the state labelled with $\tilde{k}$ represents a waiting phase (due to lock contention) preceding the k^{th} operation. We denote with $P_{W,k}$ the probability that the requested data item by the k^{th} operation is currently locked.

Finally, we represent the transaction access pattern as a $I \times M$ access matrix denoted by A. Element $A_{i,k}$ expresses the probability that the k^{th} operation of the transaction accesses the i^{th} data item. Note that the sum of each column of A must be equal to 1. Further we represent with a vector W, with $|W| = M$, the type of the access (read/write) performed by the transaction in the different phases of its execution. Specifically, W_k (resp. $1 - W_k$) is probability that the k^{th} operation is a write (resp. read) operation. Actually, the access matrix A is the building block allowing our model to capture the transaction execution history, and its effects on performance, in terms of locality variation in different phases of its execution. As an example, the matrix can be instantiated in a way that the access to a specific item j inside the transactional system is always prevented up to a given phase f of the transaction execution (this can be done by setting to 0 all the elements $A_{i,k}$ with $i = j$ and $k \leq f$). This, in its turn, captures scenarios where, e.g., items inside a given table of a database are always accessed after operations on other tables have been already executed.

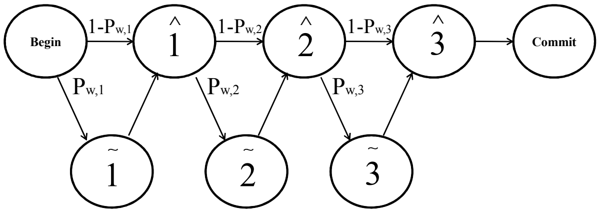

Figure 1: Transaction model.

For the sake of clarity, let us now consider an example transaction T characterized by a simple access pattern on a small database consisting solely of $I = 4$ different data items. Let us assume that T carries out 2 data accesses, respectively a read and a write operation. The read operation accesses with equal probability either data item 1 or data item 2, whereas the write operation is deterministically targeted to data item 3. Based on these assumptions, we can describe the data access pattern of T through the following access matrix A and vector W:

$$A = \begin{pmatrix} 0.5 & 0 \\ 0.5 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad W = \left| \begin{array}{c} 0 \\ 1 \end{array} \right|$$

The mean transaction response time can be evaluated as the sum of the average times spent in each state, that is:

$$R_{tx} = R_{begin} + \sum_{k=1}^{M} (\tilde{R_k} + \hat{R_k}) + R_{commit},$$

where R_{begin} and R_{commit} are times spent in states *begin* and *commit* respectively, and $\hat{R_k}$ and $\tilde{R_k}$, with $1 \leq k \leq M$, are times spent in states $\hat{k}$ and $\tilde{k}$, respectively. The evaluation of these times is presented in the following sections.

3.2.2 Lock Holding Time

The wait phase experienced by a transaction for lock acquisition on a given data item depends on the average lock

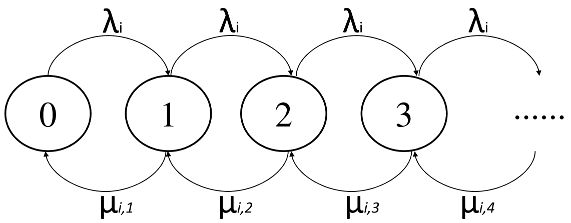

Figure 2: Markov chain for data item i.

holding times of transactions preceding T in the lock access queue on that item. In our model we explicitly capture the fact that accesses to data items can occur at different phases of a transaction. Hence, if data item a is typically the first one to be accessed by transactions, and data item b is normally the last one to be accessed, then the average lock holding time on item a will be significantly longer than the lock holding time on item b.

We evaluate the lock holding time for each data item, and how it is affected by the transaction access pattern, by exploiting the access matrix A. Specifically, if data item is accessed by a transaction at the k^{th} operation, then it gets locked up to the end of the execution of the commit phase. Hence, the lock holding time for the access to data item i at the k^{th} phase can be expressed as:

$$D_k = \sum_{j=k}^{M} \hat{R}_j + \sum_{j=k+1}^{M} \tilde{R}_j + R_{commit}.$$

We know that the probability to access data item i at the k^{th} transaction phase is expressed as $A_{i,k}$. Hence, the average lock holding time for data item i can be evaluated as:

$$Th_i = \frac{\sum_{k=1}^{M} A_{i,k} D_k}{\sum_{k=1}^{M} A_{i,k}},$$

where the sum at denominator is due to the fact that the average lock holding time must be evaluated by considering only the transactions for which an access to data item i actually occurs. We further assume that the lock holding time for each data item is exponentially distributed. Although, by construction, the expression for Th_i would determine an Erlang distribution (recall that $\hat{R}_j$ terms are assumed to be exponentially distributed in compliance with the M/M/k model for the CPU), the exponential approximation is reasonable and relatively accurate up to non-minimal values for M. This approximation will be exploited in the next section while modeling contention effects on each data item.

3.2.3 Data Contention

The arrival rate of read accesses towards the i^{th} data item can be expressed as:

$$\lambda_{read,i} = \lambda \sum_{k=1}^{M} A_{i,k}(1 - W_k),$$

while for write accesses we have:

$$\lambda_{write,i} = \lambda \sum_{k=1}^{M} A_{i,k} W_k.$$

Note that these arrivals form two Poisson processes where $\lambda_{read,i}$ and $\lambda_{write,i}$ express the corresponding mean values.

If the data item is requested by a write operation and it is currently locked (in either shared or exclusive mode), then the transaction is blocked and the write operation is enqueued. On the other hand, if the data item is requested by a read operation, the transaction is blocked only in case the item is currently locked in exclusive mode. Due to these peculiarities in the queuing policies for read and write accesses, results from the queuing theory for classical queues (e.g. the M/M/1 queue) cannot be used to model a data item as a single server sustaining the stream of interleaved read/write requests.

To cope with the determination of data contention, we have modeled the request arrival process to each single data item i as a birth-death process with fixed arrival rate, equal to

$$\lambda_i = \lambda_{read,i} + \lambda_{write,i}$$

and variable service rate $\mu_{i,j}$ (see Figure (2)), where j corresponds to the number of standing requests for data item i in the corresponding state of the Markov chain. For each single data item i the value $\mu_{i,j}$ depends on the interleaving of read and write requests observed in state j. We evaluate $\mu_{i,j}$ with its average value, calculated as follow. If in state j the top standing request for lock access is a write request, then $\mu_{i,j}$ is equal to $\frac{1}{Th_i}$. In fact, since the exclusive write lock delivered to the write request blocks any other standing request, then the item is reserved for the write request for the whole lock holding time (whose expected value is exactly Th_i). On the other hand, if the top standing request is a read request, all the other standing read requests, if any, can be concurrently served. In the latter case, if in state j there are $l \leq j$ standing read requests, then we have $\mu_{i,j} = \frac{l}{Th_i}$. In any case, we recall that Th_i has been assumed to be exponentially distributed (see Section 3.2.2), which allows solving the birth-death process via standard techniques [16]. Overall, denoting with $P_{read,i}$ and $P_{write,i}$ the probability that an incoming access request is a read request or a write request, respectively, we have

$$P_{read,i} = \frac{\lambda_{read,i}}{\lambda_i}$$

and

$$P_{write,i} = \frac{\lambda_{write,i}}{\lambda_i}.$$

We approximate the probability for the top request in state j to be a write request (respectively a read request) with $P_{write,i}$ (respectively $P_{read,i}$).

Thus, after some algebra, we have

$$\mu_{i,j} = \frac{1}{Th_i}(P_{write,i} + ((P_{read,i} \sum_{k=1}^{j-1} k P_{read,i)}^{k}) + j P_{read,i)}^{j}))$$

When a write access occurs, a conflict is raised if the target data item i is locked either in shared or in exclusive mode. Thus we can model the contention probability for a write access $PW_{write,i}$ on data item i as the sum of the probabilities to stay in any of the states j, with $j > 0$, of the Markov chain, which is equal to $1 - P_0$ (where P_0 is the probability to be in state 0 of the Markov chain). Hence, from queuing theory [16], we have

$$PW_{write,i} = 1 - \frac{1}{1 + \sum_{k=1}^{\infty} \prod_{j=0}^{k-1} \frac{\lambda}{\mu}_{i,j+1}},$$

By the formula it can be noted that $PW_{write,i} \leq 1$ only if the sum at the denominator converges to a finite value. Given that $\mu_{i,j} \geq \frac{1}{Th_i}$ $\forall j > 0$, the condition $\frac{1}{Th_i} > \lambda_i$ for every data item i in the transactional system is sufficient to ensure that the contention probability for any write access is less than 1, thus representing a stability condition for the system.

To evaluate the contention probability of a read access we recall that a conflict can occur only if the data item is locked in exclusive mode. Hence, the contention probability can be evaluated as the fraction of time during which the data item is locked in exclusive mode. This time fraction corresponds to the utilization of the data item vs write accesses. Thus we have

$$PW_{read,i} = \lambda_{write,i} Th_i.$$

3.2.4 Wait Time

When an incompatible lock is found on the currently required data item, the transaction experiences a wait time t, which corresponds to the time spent in state $\tilde{k}$ (see Figure 1), with k being the index of the operation causing the conflicting access. The wait time depends on the data being requested (i.e. on the amount of currently standing access requests for that data item), not on the value of k.

We firstly evaluate the average waiting time for a transaction in case of a conflict on a specific data item i, which we denote as $R_{wait,i}$. After we evaluate the average waiting time experienced in each state $\tilde{k}$ (with $1 \leq k \leq M$), which we denote as $\tilde{R_k}$. The latter value will depend on the transaction access matrix A, which expresses, for each operation, the likelihood of access to each specific item.

$R_{wait,i}$ can be evaluated through the aforementioned Markov chain associated with data item i. In particular, the average amount of standing accesses is

$$N_i = \sum_{j=1}^{\infty} j P_j,$$

where P_j is probability to stay in state j of the Markov chain. When a conflict occurs upon data access, if no other access requests to the same item are currently queued, the wait time corresponds to residual lock holding time. On the other hand, in case other access requests are currently queued for lock acquisition, a further delay occurs due to lock holding on that item by transactions associated with the queued request, that is on average Th_i for each one. Thus we have $R_{wait,i} = (N_i - 1)Th_i + L_i$, where:

$$L_i = \frac{\sum_{k=1}^{M} A_{i,k} D_k^2 0.5}{\sum_{k=1}^{M} A_{i,k} D_k}$$

and represents the normalized residual lock duration, depending on the different durations evaluated on the basis of the access pattern. Now, through $R_{wait,i}$, by exploiting the access matrix A, we have

$$\tilde{R_k} = \sum_{i=1}^{I} \sum_{k=1}^{M} A_{i,k} R_{wait,i} (PW_{read,i} P_{read,i} + PW_{write,i} P_{write,i})$$

3.2.5 Operations Execution Time

Times spent by a transaction in states *begin*, $\hat{k}$, with $1 \leq k \leq M$, and *commit* can be evaluated by exploiting the model of the underlying hardware resources, which has been provided at the beginning of Section 3.1. The CPU load for the execution of a transaction is

$$C_{cpu} = nIST_b + M \cdot nIST_o + nIST_c$$

and from queuing theory we get for the CPU utilization the following expression:

$$\rho = \frac{\lambda \cdot C_{cpu}}{k \cdot MIPS}.$$

Denoting with $p[queuing]$ the wait probability for a request in an M/M/k queue [16], and defining γ as

$$\gamma = 1 + p[queuing]/(k(1-\rho)),$$

we can evaluate the response times R_{begin} and R_{commit} of states *begin* and *commit* respectively as:

$$R_{begin} = \gamma \frac{nIST_b}{MIPS}$$

and

$$R_{commit} = \gamma \frac{nIST_c}{MIPS}.$$

Response times of states $\hat{R_k}$ further depend on buffer hit probability and I/O delays. Using the notation in Section 3.1, we have

$$\hat{R_k} = \gamma \frac{nIST_o}{MIPS} + P_{BH} \cdot t_{I/O}$$

for each k such that $1 \leq k \leq M$.

3.2.6 Numerical Resolution

The model can be solved via an iterative procedure. After assign the values to hardware configuration parameters (e.g the CPU power) and transactional system parameters (e.g. the access matrix) the value 0 has to be assigned to the parameters $R_{wait,i}$, $PW_{read,i}$ and $PW_{write,i}$ (with $1 \leq i \leq I$). Then the other model parameters can be evaluated via the provided equations, using the results as the input for the next iteration. The desired computational accuracy can be fixed by defining a value ϵ specifying the maximum difference between values obtained by two consecutive iterations (e.g. if R_n is the transaction response time at iteration n, then the computation can be stopped when the condition $R_n - R_{n-1} \leq \epsilon$ becomes true). While it is out of the scope of this paper to formally demonstrate the convergence of this iterative solution method (which has been adopted for solving several pre-existing performance models of concurrency control protocols, see, e.g, [6, 7, 13, 15, 26]), we have empirically observed that it always converges in a few iterations, provided that the input assignment defines a stable system.

3.3 Coping with Multiple Transaction Classes

In this subsection we show how our model can be employed in scenarios where the workload entails different transaction profiles (or classes). We denote as C the number of the different transactional classes, each of which can be represented through a specific transaction model featured as the one described in 3.2.1. We use the following notation:

- Vector $\overline{M}$, with $|\overline{M}| = C$, where element $\overline{M}^c$, with $1 \leq c \leq C$, represents the number of operations of a transaction of class c.

- Vector $\overline{A}$ of matrix elements, with $|\overline{A}| = C$, where element $\overline{A}^c$, with $1 \leq c \leq C$, is the access matrix of a transaction of class c.

- Vector $\overline{W}$, with $|\overline{W}| = C$, where element $\overline{W}^c$, with $1 \leq c \leq C$, is the write probability for the m^{th} operation of a transaction of class c.

Further the transaction arrival rate for class c is denoted by λ_c.

The accuracy level while describing a workload with differentiated transaction profiles according to the previous notation can be tuned in accordance to the requirements of the performance analysis the end-user is carrying out. Roughly speaking, the more the identified transactional classes, the more accurate the workload description. As an extreme, each plausible access pattern could be associated with a specific class in such a way to describe the variation of the transaction locality over the data items in a deterministic manner. In this case each access matrix will be characterized by columns having a single element equal to 1, and all the other elements equal to 0. As it will be clear by the below description of the modifications to the model equations in case of multiple classes, a large number of classes will only entail an increased amount of computation power for the iterative model solving procedure. In general, if transactions are composed by a fixed number of predefined statements, as in, e.g., a lot of three-tier Web based applications, to obtain a good compromise we suggest to model the workload using a single class for each predefined transaction pattern.

With more transaction classes, some of the previously introduced equations must be rewritten in order to consider parameter dependency on the access pattern and the arrival rate of each class. For simplicity, we only show the final shape of these equations without explicitly repeating intermediate modeling steps, which are anyway intuitive once the base model in Section 3.2 has been analyzed. The transaction response time for class c is

$$R_{tx}^c = R_{begin}^c + \sum_{k=1}^{M} (\tilde{R_k^c} + \hat{R_k^c}) + R_{commit}^c,$$

where we added the superscript c to the parameters introduced in Section 3.2.1 to emphasize that each of them is related to class c. The average lock holding time for data item i, $\tilde{R_k^i}$, becomes

$$\frac{\sum_{c=1}^{C} \lambda^c A_{i,k}^c (\sum_{j=k}^{M^c} \hat{R}^c{}_j + \sum_{j=k+1}^{M^c} \tilde{R}^c{}_j + R_{commit}^c)}{\sum_{c=1}^{C} \lambda^c \sum_{k=1}^{M^c} A_{i,k}^c},$$

The arrival rates of read and write accesses towards the i^{th} data item become

$$\lambda_{read,i} = \sum_{c=1}^{C} \lambda^c \sum_{k=1}^{M^c} A_{i,k}^c (1 - W_k^c),$$

and

$$\lambda_{write,i} = \sum_{c=1}^{C} \lambda^c \sum_{k=1}^{M^c} A_{i,k}^c (W_k^c).$$

#Items	1000
#CPUs	5
CPU Speed	10 MIPS
$t_{I/O}$	0.035s
#Accesses x Xact (M)	15
P_{write}	100%
P_{BH}	0.27
$nIST_b$	150000
$nIST_o$	20000
$nIST_c$	250000
Access Distribution - Phase i (Phase Independent Workload)	Unif. in [1,1000]
Access Distribution - Phase i (Phase Dependent Workload)	Unif. in $[1+\lfloor \frac{i-1}{3} \rfloor \cdot 200, (\lfloor \frac{i-1}{3} \rfloor + 1) \cdot 200]$

Table 1: Parameters settings for Part-A (as in [26]).

In the end, we can rewrite the equation of $\tilde{R_k}$ for each transaction class as

$$\tilde{R_k^c} = \sum_{i=1}^{I} \sum_{k=1}^{M^c} A_{i,k}^c R_{wait,i} (PW_{read,i} P_{read,i} + PW_{write,i} P_{write,i})$$

4. MODEL VALIDATION

In this section we present a validation study of the proposed analytical model. We evaluate the model accuracy via a set of differentiated tests based on output comparison vs the performance of SS2PL as obtained by a discrete event simulator developed using the `C` programming language. To ensure reproducibility of simulation results, the simulator code has been made publicly available. We made as well freely available the Java-based implementation of our analytical model solver (see [1] for both packages). The latter package may reveal a valuable tool for performance engineers and database administrators in charge of carrying out capacity planning of lock-based transactional systems.

As the last preliminary observation, the simulation software explicitly implements data accesses and lock management algorithms proper of the SS2PL concurrency control protocol, while it simulates the underlying hardware components by modeling them via the corresponding queuing systems. Transaction deadlocks possibly arising due to lock contention are resolved in the simulator immediately upon the generation of a cycle within the transaction wait-for-graph [10]. Deadlock resolution results in aborting the transaction causing deadlock, which is immediately resubmitted for a new execution.

The tests are related to three main scenarios characterized by diverse workload configurations and system parameters. In particular, this section is structured as follow. In **Part-A** we validate our model by using the same parameters' configuration used in [26]. This will also permit a direct comparison between the accuracy of our model and of the one presented in that same work. In **Part-B** we consider synthetic workloads which induce worst case effects on lock contention across different transaction classes. This has been done to stress the robustness of the model in the presence of diverse transaction access patterns. In **Part-C** we finally provide validation results for the case of transaction workloads derived by abstracting the main features of the well known TPC-C benchmark [24].

#Items	100000
#CPUs	8
CPU Speed	2000 MIPS
$t_{I/O}$	0.004 ms
P_{BH}	0
P_{write}	20%

Table 2: Parameters settings for Part-B.

4.1 Part-A

We start validating our analytical model by considering the same parameters setting (reported in Table 1) that has been used to evaluate the accuracy of the SS2PL model presented in [26]. According to this setting, we consider two different transaction profiles, both entailing 15 data accesses in write mode. In the first transaction profile (which we name phase-independent), the accesses are uniformly distributed across the whole set of items in the transactional system, independently of the transaction execution phase. In the second transaction profile (which we name phase-dependent), data accesses are still uniformly distributed (when considering the transaction as a whole). However, the data access locality varies across different transaction execution phases. Specifically, for this transaction profile we see the items inside the transactional systems as logically partitioned into 5 equally sized, non-overlapping sets $\{S_1, \dots, S_5\}$ (which might be seen as representative of, e.g., distinct database tables). The transactions perform three accesses, uniformly distributed in the set S_1, and then sequentially move to the next sets (with 3 accesses in each set), until they complete the pre-established number of 15 data accesses.

In Figure 3 we plot, for both the phase-independent and phase-dependent transaction profiles, the average transaction execution time as evaluated by (i) our analytical model, (ii) the analytical model in [26], and (iii) the simulator, used as the reference for estimating the accuracy of the two analytical models. For the phase-independent transaction profile, it can be noted that the SS2PL model proposed in this paper is more accurate than the one in [26], especially when the probability of data conflict grows (i.e. at high workload). In particular, our model well captures system saturation, while the model in [26] largely underestimates the system saturation point, with an error of about 60%.

This depends on that, differently from our approach, the model in [26] determines the average lock waiting time by exploiting a number of simplifying assumptions (e.g. limiting the lock queue length to 1 and then compensating by upper bounding the data access conflict probability) that introduce larger errors in scenarios characterized by high conflict rates.

When moving from the phase-independent workload to the phase-dependent workload with the same popularity for each data item, it can be noted that the system performance gets remarkably affected. Given that the effects of access locality variations across different transaction execution phases are not captured by the model in [26], it would lead to the same curve seen for the phase-independent case and exhibit a significant discrepancy with respect to the simulator output. Conversely, our model, which explicitly captures this phenomenon, provides highly accurate performance predictions even for the phase-dependent case.

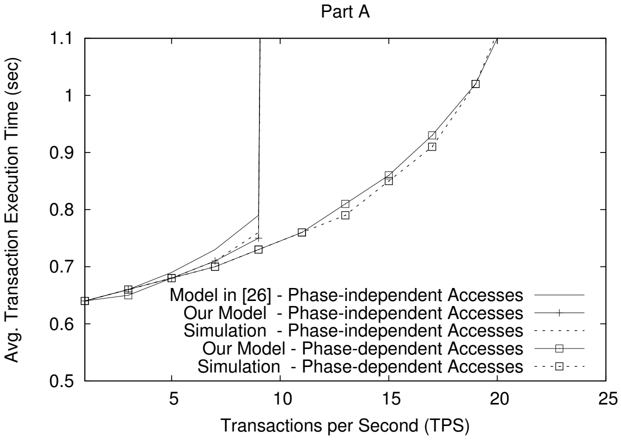

Figure 3: Performance comparison for both independent and phase-dependent data access patterns (Part-A).

	#Accesses (M)	Access Distribution - Phase i ∈[1,M]
Profile P_1	20	Uniform in $[1+\lfloor\frac{i-1}{4}\rfloor\cdot 20000,(\lfloor\frac{i-1}{4}\rfloor+1)\cdot 20000]$
Profile P_2	8	Uniform in $[1+\lfloor\frac{i-1}{4}\rfloor\cdot 20000,(\lfloor\frac{i-1}{4}\rfloor+1)\cdot 20000]$
Profile P_3	8	Uniform in $[1+(\lfloor\frac{i-1}{4}\rfloor+3)\cdot 20000,(\lfloor\frac{i-1}{4}\rfloor+4)\cdot 20000]$

Table 3: Synthetic workload 1 (Part-B).

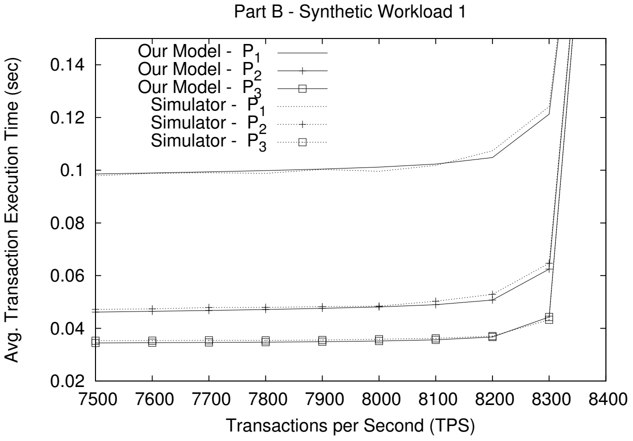

Figure 4: Transaction execution time for synthetic workload 1 (Part-B).

#Accesses (M)	15
Profile P_1 Access Distribution - Phase i	Uniform in $[1+\lfloor\frac{i-1}{5}\rfloor\cdot 200,(\lfloor\frac{i-1}{5}\rfloor+1)\cdot 200]$
Profile P_2 Access Distribution - Phase i	Uniform in $[1+\lfloor\frac{M-i-1}{5}\rfloor\cdot 200,(\lfloor\frac{M-i-1}{5}\rfloor+1)\cdot 200]$

Table 4: Synthetic workload 2 (Part-B).

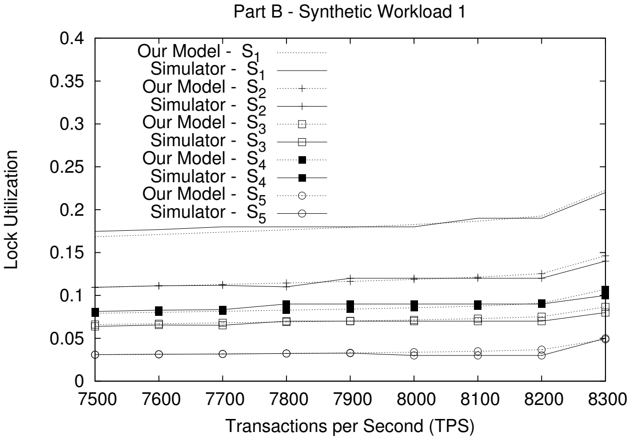

Figure 5: Lock utilization for synthetic workload 1 (Part-B).

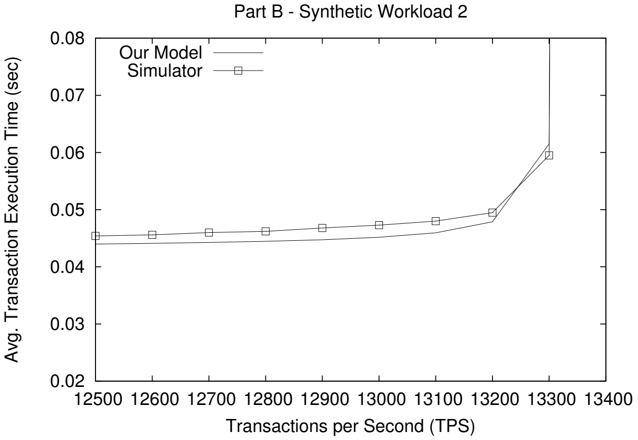

Figure 6: Transaction execution time for synthetic workload 2 (Part-B).

4.2 Part-B

We now focus our experimental study on the evaluation of the accuracy of our model in more complex scenarios characterized by multiple transaction profiles and highly skewed phase-dependent data access distributions. Further, compared to the study in the previous section, we consider system parameters representative of more modern platforms (e.g. an increased number of CPUs/Cores and increased processor/disk speed) and applications (e.g. an increased amount of items inside the transactional system). The detailed parameter settings adopted for this study are reported in Table 2. As the last preliminary consideration, this time the value of P_{BH} (which would depend on the specific object replacement policy) has been set to 0. (Recall that our analysis is orthogonal to modeling approaches for buffer replacement policies and related hit/miss effects vs the item popularity.)

We analyze two different synthetic workloads which only share the following two characteristics. Data items are grouped in 5 contiguous sets (logically equivalent to, e.g., database tables) which we again refer to as $\{S_1, \dots, S_5\}$. Also, the probability of access in write mode is set equal to 20%. Synthetic workload 1 (see Table 3) entails three different transaction profiles P_1, P_2 and P_3, with identical arrival rates, and the following access patterns. For class P_1, the pattern is similar to the phase-dependent pattern of Part-A of our study, with the only variation that the number of accesses is equal to 20, and 4 accesses per set are executed before moving to the subsequent set. Transactions of class P_2 perform 4 accesses to the set S_1 and then other 4 accesses to the set S_2 (for a total of 8 accessed items). Similarly, transactions of profile P_3 perform 4 accesses to the set S_4, and 4 subsequent accesses to the set S_5. In every transaction profile, the 4 accesses in each set are uniformly distributed over the whole items in that set. The performance plots for this workload (see Figure 4) show a very good matching between simulation and analytical values for all the three transaction classes. Also, despite the transaction profiles P_2 and P_3 access the same number of items having the same global popularity, the relative response times differ by about the 35%. This phenomenon is due to the presence of profile P_1, which locks the items of sets S_1 and S_2 (also accessed in profile P_2) for longer time intervals than the items in sets S_4 and S_5 (also accessed by profile P_3). For this same workload, we also show (see Figure 5) a comparison between the lock utilization values for each of the 5 sets as predicted by both the simulation and the analytical model. Beyond confirming the tight matching between simulation and analytical results, these plots highlight an interesting feature of our model. Specifically, its ability to capture data contention dynamics with single data item granularity makes it capable to predict the performance effects due to the specific organization of the transactional logic (such as the order of the accesses to different data sets within different phases of a transaction). As an example, Figure 5 highlights that the accesses to the set of items S_1 represent the system bottleneck.

The second synthetic workload (number 2) we consider represents a “stress” case aimed at assessing the accuracy of our analytical model (see Table 4). This workload consists of two transaction profiles, denoted as P_1 and P_2, with identical length, 15 accesses, and arrival rates. The data accesses for the two profiles are symmetric. Specifically, transactions of profile P_1 perform three accesses to each set S_i starting from S_1 and then sequentially moving according to increasing set indexes. Instead, transactions of profile P_2 perform three accesses to each set starting from S_5 and then move to the other sets according to a (reverse) decreasing order of the set indexes. In such a configuration, items in the sets with extreme indexes (i.e. index 1 and index 5) experience lock holding times with high variance across the two transaction profiles, which is the main contribution towards the stress configuration for Mean-Value-Analysis (MVA) approaches like our one. Also for this workload, the plotted curves (see Figure 6) show a very good matching between simulation and analytical performance values ([2]).

4.3 Part-C

We conclude our experimental study by evaluating the accuracy of our model with a workload reflecting relevant features of a standard benchmark for transactional systems,

[2] Due to symmetry, the response time for the two transaction profiles is the same, which is the reason why we only plotted a single analytical/simulated curve.

Table Name	# Items	Table ID
WAREHOUSE	500	tb0
DISTRICT	1000	tb1
CUSTOMER	15000	tb2
STOCK	500000	tb3
ITEM	100000	tb4
ORDER	1000	tb5
NEW-ORDER	1000	tb6
ORDER-LINE	1000	tb7
HISTORY	1000	tb8

Table 5: TPC-C tables' population.

Phase	P_0 (47%)	P_1 (45%)	P_2 (4%)	P_3 (4%)
0	(R),tb0	(R),tb0	(R),tb2	(R),tb6
1	(R),tb1	(R),tb1	(R),tb5	(W),tb6
2	(W),tb1	(R),tb2	(R),tb7	(R),tb5
3	(R),tb2	(W),tb0		(W),tb5
4	(W),tb5	(W),tb1		(R),tb7
5	(W),tb6	(W),tb2		(W),tb7
6	(R),tb4	(W),tb8		(R),tb2
7	(R),tb3			(W),tb2
8	(W),tb3			
9	(W),tb6			

Table 6: Abstracted TPC-C transaction profiles (classes).

namely TPC-C [24]. The item tables' population and layout (see Table 5) have been configured by setting the number of warehouses (which represent an explicit scale parameter for TPC-C) to 500. The only variation is related to the scaling of the size of the tables which are accessed via select statements using intervals of keys. This choice is motivated by the fact that such select statements would lead to K read operations, as modeled in our approach. Therefore the scaling has been done in order to provide a fair modeling approach for select (i.e. read) statements operating at different granularity values (single key vs interval of keys).

The characterization of the transactional data access patterns is based on the TPC-C workload modeling carried out in [17]. Table 6 reports, for each transaction profile and transaction execution phase, which item table is accessed as well as the corresponding access mode (read, denoted as (r), vs write, denoted as (w)). We consider only 4 of the 5 different transactional classes identified in [17], since one of them, namely the Stock-level transaction, does not impose any isolation guarantee, hence not triggering any concurrency control mechanism at all (whose modeling is the focus of this work). The remaining model parameters (characterizing, e.g., the available hardware resources) are not reported as they are unchanged with respect to Section 4.2.

By the results in Figure 7, it can be observed that our model well fits the simulation output. As for previous cases, the matching can be observed for each single transaction profile included in the workload. These results confirm the high accuracy of our analytical performance model even in case of complex and diverse workloads.

5. ASSESSMENTS AND CONCLUSIONS

In this work we have proposed a novel analytical model for one of the most widely employed concurrency control schemes, namely (Strong Strict) 2-Phase-Locking.

Our solution overcomes several limitations of preexisting analytical approaches, such as the simplifying assumption that accesses across different execution phases of a transaction are identically distributed. Through an extensive experimental study, we have shown that our model is capable of achieving significantly a higher level of accuracy with respect to existing methods for a wide variety of synthetic and realistic workloads.

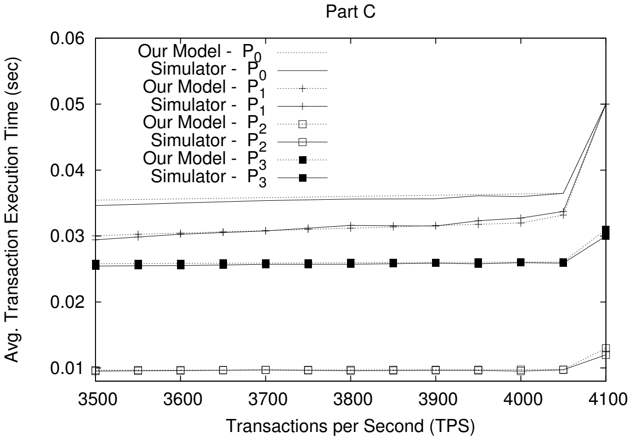

Figure 7: Simulation and analytical results for the abstracted TPC-C workload.

Our analytical model takes as input parameter a so called access matrix, which encodes the transaction access pattern associated with data items. Planned future work encompasses the study of methods for automatic generation of the access matrix in generic transactional contexts.

6. REFERENCES

[1] http://www.dis.uniroma1.it/pub/quaglia/2pl-simulator-plus-modelsolver.tar.gz.

[2] K. A. Merchant, P. Yu, and M. Chen. Performance analysis of dynamic finite versioning for concurrent transaction and query processing. *ACM SIGMETRICS Performance Evaluation Review*, 20(1), June 1992.

[3] R. Agrawal, M. J. Carey, and M. Livny. Concurrency control performance modeling: Alternatives and implications. *ACM Transactions on Database Systems*, 12(4), December 1987.

[4] N. Al-Jumaha, H. Hassaneinb, and M. El-Sharkawia. Implementation and modeling of two-phase locking concurrency. *Information and Software Technology*, 42(4):257–273, March 2000. Elsevier Science.

[5] R. Balter, P. Berard, and P. Decitre. Why control of the concurrency level in distributed systems is more fundamental than deadlock management. In *Proceedings of the First ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Canada*, pages 183–193. ACM New York, NY, USA, 1982.

[6] B.Ciciani, D.M.Dias, and P.S.Yu. Analysis of concurrency-coherency control protocols for distributed transaction processing systems with regional locality. *IEEE Transactions on Software Engineering*, Volume 18(10):pp. 899–914, October 1992.

[7] B.Ciciani, D.M.Dias, and P.S.Yu. Dynamic and static load sharing in hybrid distributed-centralized systems. *Computer Systems Science and Engineering*, Volume 7(1):pp. 25–41, January 1992.

[8] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, volume 99, pages 1–10, May 22-25 1995. San Jose, California, United States.

[9] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[10] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publishing Co., Inc., 1987.

[11] M. J. Carey and W. A. Muhanna. The performance of multiversion concurrency control algorithms. *ACM Transactions on Computer Systems*, 4(4):338–378, November 1986.

[12] B. Ciciani, F. Calderoni, A. Santoro, and F. Quaglia. Modeling of QoS-oriented content delivery networks. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, pages 341–344, Washington, DC, USA, 2005. IEEE Computer Society.

[13] P. di Sanzo, B. Ciciani, F. Quaglia, and P. Romano. A performance model of multi-version concurrency control. In *Proceedings of the 16th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2008)*, pages 41–50. IEEE Computer Society, 2008.

[14] J. Gray, P. Homan, R. Obermarck, and H. Korth. A straw man analysis of probability of waiting and deadlock. *IBM Research Report RJ 3066*, 1981.

[15] I.K.Ryu and A.Thomasian. Analysis of database performance with dynamic locking. *Journal of the ACM (JACM)*, Volume 37(Issue 3):pp. 491 – 523, July 1990.

[16] L. Kleinrock. *Queuing Systems (Vol1 and Vol2)*. Wiley-Interscience, 1975.

[17] S. T. Leutenegger and D. Dias. A modeling study of the tpc-c benchmark. *SIGMOD Rec.*, 22(2):22–31, 1993.

[18] D. R. Ries and M. Stonebraker. Locking granularity revisited. *ACM Transactions on Database Systems (TODS)*, 4(2), 1974.

[19] D. R. Ries and M. Stonebraker. Effects of locking granularity in a database management system. *ACM Transactions on Database Systems (TODS)*, 2(3), September 1977.

[20] A. Thomasian. On a more realistic lock contention model and its analysis. In *Proceedings of the 10th International Conference on Data Engineering*, pages 2–9, Feb 1994.

[21] A. Thomasian. A more realistic locking model and its analysis. *Information Systems*, 21(5):409–430, 1996.

[22] A. Thomasian. Concurrency control: Methods, performance, and analysis. *ACM Computing Surveys*, 30(1), March 1998.

[23] A. Thomasian and I. Ryu. Performance analysis of two-phase locking. *IEEE Transactions on Software Engineering*, Volume 17(Issue 5):386 – 402, May 1991.

[24] Transaction Processing Performance Council. *TPC Benchmark^{TM} C, Standard Specification, Revision 5.1*. Transaction Processing Perfomance Council, 2002.

[25] P. Yu and M. Chen. Performance analysis of dynamic finite versioning schemes: storage cost vs. obsolescence. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), December 1996.

[26] P. S. Yu, D. M. Dias, and S. S. Lavenberg. On the analytical modeling of database concurrency control. *Journal of the ACM (JACM)*, 40(4):831–872, September 1993.