On Exploring Markov Chains for Transaction Scheduling Optimization in Transactional Memory

Pierangelo Di Sanzo, Marco Sannicandro,
Bruno Ciciani, Francesco Quaglia

Abstract

Software Transactional Memory (STM) systems' performance can be strongly affected by the way transaction parallelism, and its actual degree, is reflected into conflicting data accesses. We illustrate an approach for STM performance prediction, which is based on exploiting Markov Chain formalisms to determine the effects on performance (particularly on throughput) by the scheduling of STM transactions. We focus on the scenario where the scheduling rule relies on either temporarily blocking or not an issued transaction depending on the current number of already running transactions. Our model can be used for optimization purposes, e.g., via adaptive scheduling policies aimed at dynamically changing the maximum number of transactions admitted for concurrent execution. Experimental data are also provided for validating our performance prediction approach.

1 Introduction

In Software Transactional Memory (STM) systems, performance can be strongly affected by the conflict (hence abort) rate of transactions running in parallel. This issue can be addressed using either a *transaction scheduler* [1] or a *thread scheduler* [2], which is in charge of temporarily blocking a transaction/thread depending on some specific conditions. Scheduling policies can exploit either system performance prediction schemes or heuristics. The approach proposed in [3] focuses on a performance prediction schemes based on measuring the speed-up of an STM application running with different numbers of threads. Then, a performance prediction function is instantiated by interpolating collected measurements. In [2], the authors devise a machine learning-based approach for predicting the scalability of an STM application as a function of the number of running threads. An analytical model-based approach has been described in [4]. Examples of heuristic-based schemes include the following ones. In [5], the authors propose a hill-climbing heuristic scheme, which dynamically increases or decreases the number of concurrent threads. Some heuristic-based pro-active scheduling schemes have been described in [6, 1, 7]. Basically, these schemes delay the execution of some transaction when the conflict probability is estimated to be high. Specifically, the work in [6] presents a control algorithm that dynamically changes the number of threads executing transactions on the basis of the observed transaction conflict rate. In [1], transactions are serialized when the conflict rate exceeds a given threshold. In [7], a specific transaction is serialized when its probability of conflicting with already running transactions is predicted to be higher than a given threshold value.

However, the presented approaches show some drawbacks. Performance model-based approaches, such as the ones presented in [2, 3, 4], require a priory analysis phases, during which an STM application has to be observed while running with different workload profiles and/or different system configurations (in terms of, e.g., number of concurrent transactions/threads). Heuristic-based schemes, such as [6, 1, 7, 5], require the user to configure various parameters, such as conflict rate thresholds, or transaction abort probability thresholds, based on which the scheduler takes decisions. We note that, there are no conflict rate values and/or transaction abort values for which all STM applications can be expected to provide the best throughput. Additionally, since the workload profile of an application can change during the application execution, specific values of some thresholds could be optimal for some phases of the application execution, while they could be unsuitable for other phases.

To cope with the above-mentioned issues, we developed an effective and lightweight performance prediction approach, which exploits a Markov Chain-based [8] performance model for STM systems. With our approach, we aim at enabling transaction scheduling *what-if* analysis by providing a performance model whose input parameters can be measured on-the-fly while observing an STM system running. Particularly, by observing the

system running with a fixed transaction scheduling configuration, in terms of maximum number of admitted concurrent transactions, our scheme can predict the system throughput if the scheduling configuration would change (i.e. lower or higher transaction concurrency levels would be allowed). The scheme does not uses any configuration parameter that needs to be set by the user, except the length of the time window during which the system has to be observed (so that the model can be instantiated). This length can simply be expressed in terms of number of consecutive committed transactions. Ultimately, our scheme can be easily used for run-time optimization of transaction scheduling. We show the feasibility of our approach via an experimental study that we conducted using a real instantiation of an STM system with a benchmark application.

We illustrate our modelling approach in Section 2 and we present the results of the experimental (validation) study in Section 3.

2 Markov Chain-based STM Performance Model

In this section, we present the Markov Chain-based performance model of the STM system. We initially provide a description of the target STM environment, thus providing the reference system model for our analysis, then we focus on the derivation of the performance model.

2.1 Target STM Model

We assume an STM system where a number N of concurrent threads are run. A thread can execute either transactional code or non-transactional code (*ntc*) blocks. A transaction commits if no conflicts with other concurrent transactions occur, otherwise, it is aborted and a new run of the same transaction is executed. Upon the start of a new transaction along any thread, it gets blocked by the transaction scheduling logic (namely it enters a *waiting* state) in case there are m transactions (with $m \leq N$) that are already within their running phase. When one of the m running transaction commits, a waiting transaction, if any, is unblocked by the scheduler, thus being allowed to proceed along its execution path.

2.2 Performance Model Derivation

Our performance model of the STM system leverages a Continuous Time Markov Chain (CTMC) [8] with N states. A state k of the CTMC represents a system state when there are k threads executing transactions, where k accounts for both already running and blocked transactions. Consequently, when the system resides in state k there are $N - k$ threads executing *ntc* blocks. A transition from state k to $k + 1$ occurs upon the startup of a transaction along any thread. A transition from the state k to the state $k - 1$ occurs upon the commit of whichever running transaction. We denote the average time for executing some *ntc* block as t_{ntc}. Thus, the transaction inter-arrival rate along any thread is $\lambda = \frac{1}{t_{ntc}}$. Consequently, denoting with λ_k the transition rate from a state k to the state $k + 1$, we have

$$\lambda_k = (N - k) \cdot \lambda \tag{1}$$

As for the transition rate from a state k to the state $k - 1$, it depends on k and m (we remark that m represents the maximum number of concurrent transactions allowed to run – hence not being blocked – by the scheduler). Denoting with t_k the average transaction execution time when there are k executing transactions, the transaction execution rate in the state k is equal to $\mu_k = \frac{1}{t_k}$. Accordingly, for any states $k \leq m$, since exactly k transactions are running (i.e. none is blocked), the transition rate from the state k to the state $k - 1$ is

$$\gamma_k = k \cdot \mu_k \tag{2}$$

Conversely, for any state $k > m$, the transactions actually running are m, while the remaining $k - m$ transactions are blocked. Consequently, for all states such that $k > m$, the transition rate from the state k to the state $k - 1$ is

$$\gamma_k = m \cdot \mu_k \tag{3}$$

The CTMC implementing this model is represented in Figure 1. Clearly, the underlying implicit assumption allowing us to rely on Markov Chains is that the parameters used in the equations, which express latency values related to state transitions (e.g. the latency for executing some *ntc* block) are exponentially distributed.

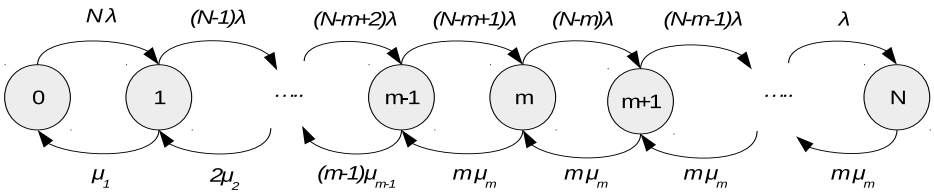

Figure 1: Markov Chain representation of the evolution of the STM system state.

Transaction execution time. Now we focus on the average transaction execution time t_k. We note that t_k is affected by the number of times a transaction is aborted (i.e. re-started) in the state k before successfully committing. We refer to as *wasted time*, which we denote as $w_{t,k}$, the average time to execute all aborted runs of a transaction (including the time to execute the abort operations) in the state k. Further, we refer to as *useful time*, which we denote as $u_{t,k}$, the average time to execute the last run of a transaction (i.e. the successfully committing run). Thus, $t_k = w_{t,k} + u_{t,k}$. Further, we note that the wasted time $w_{t,k}$ is equal to the product between the average time $w^r_{t,k}$ to execute a transaction run that gets aborted and the average number of times r_k a transaction gets aborted, i.e.:

$$w_{t,k} = w^r_{t,k} \cdot r_k \tag{4}$$

Assuming that, for a given state k, an abort event of a transaction is independent of previous abort events of the same transaction, the probability distribution of the number of runs of a transaction before successfully committing follows a geometric distribution. Thus, if p_k is the transaction abort probability in the state k, we have

$$r_k = \frac{p_k}{1 - p_k} \tag{5}$$

As a final observation, we note that we can assume that all abort probabilities p_k for $k > m$ are equal to p_m, because in all states with $k > m$, there are always m running transactions.

Throughput. Based on the CTMC that we described above, the system throughput thr_m when the scheduler admits at most m running transactions can be estimated through the CTMC stationary probability distribution. Specifically, if q_k is the stationary probability of the state k, we have

$$thr_m = \sum_{i=1}^{N} q_k \cdot \gamma_k \tag{6}$$

3 Experimental Evaluation

In this section, we present some experimental results for evaluating the accuracy of our performance model. Additionally, we illustrate how the model can be used to perform what-if analysis while varying the maximum number of transactions admitted to concurrently run by the scheduler. We executed experiments using applications of the STAMP benchmark[9] running on TinySTM [10]. Due to space constraints, we only show results for the *Intruder* application. The experiments have been executed running the application with 8 threads on a 8-cores HP ProLiant server, equipped with 2GHz AMD Opteron 6128 processors, 64 GB of RAM and the Linux operating system (kernel version 2.7.32-5-amd64).

Model accuracy. During the execution of the application, periodically (each 1000 executed transactions) the following measurements were taken for each k: 1) the average useful time of a transaction ($u_{t,k}$), 2) the average time to execute an aborted run of a transaction ($w^r_{t,k}$), 3) the abort probability of a transaction (p_k) and 4) the average time for executing a *ntc* block (t_{ntc}). After each measurement period, the throughput of the application has been estimated via the presented performance model. In the left graph of Figure 2, we show a comparison between the estimated throughput and the measured throughput along the whole execution of the application while the transaction scheduler was admitting at most 4 running transactions (i.e. $m = 4$). The average relative error we observed is equal to 8.9%.

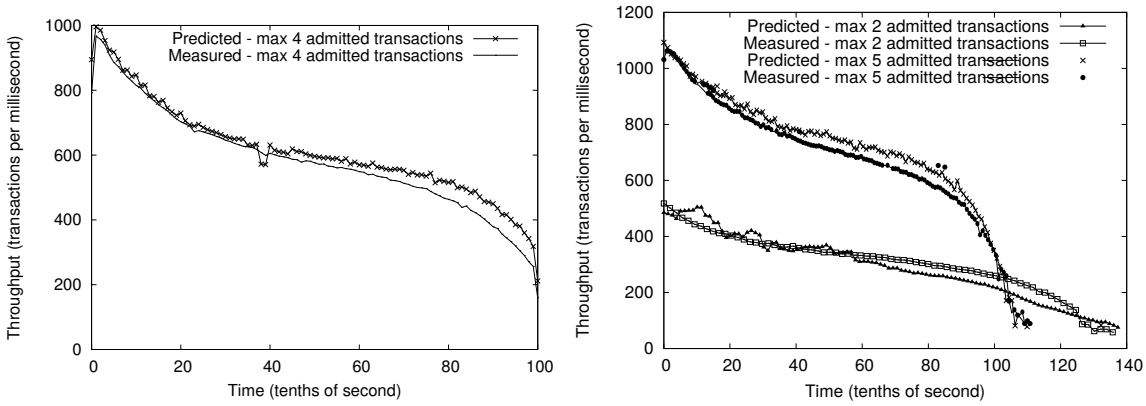

Figure 2: Predicted and measured throughput for different configurations of the scheduler.

What-if analysis. Now, we show how the proposed model, using measurements taken while running the application with a given number of admitted transactions, say x, can be used for predicting the system throughput when a different number of transactions are admitted, say x'. By construction of the model, this can be done using the values of parameters $u_{t,k}$, $w^r_{t,k}$, p_k and t_{ntc} measured when running the application with x admitted transactions, and setting $m = x'$ in all model equations. However, we note that, for predicting the system throughput in the cases where $x' > x$, the transaction abort probabilities $\{p_k : x < k \leq x'\}$ have to be estimated (in fact, they can not be measured if the system has been observed when admitting no more then x transactions). We describe in the appendix the approach that we used for calculating the above set of abort probabilities.

We used measurements taken while running the application with $m = 4$ to predict, for each measurement period, the throughput for $m = 2$ and $m = 5$, respectively. In the right graph of Figure 2, we plot the predicted throughput and the measured throughput. The average prediction error that we observed was 8.8% for $m = 2$ and 9.2% for $m = 5$. These results support the accuracy of our analysis. They also indicate that the presented model could be effectively used at run-time for tuning the scheduler configuration. This could be done by periodically estimating the value of m for which the predicted throughput is maximized. We plain to explore this possibility in future work.

4 Conclusions

We presented a Markov Chain-based performance model for STM systems. The model aims at estimating the system throughput in presence of a scheduler which limits the number of admitted running transactions in order to prevent performance degradation. Particularly, the model can be instantiated on-the-fly and does not require the user to configure any parameter, except the length of the system observation window. We provided some experimental results for evaluating the model accuracy. Also, we showed that, by observing the system running with a given number of admitted transactions, the model can be used to perform what-if analysis while changing such a number. Experimental results show that the proposed modeling approach is likely suited to support run-time configuration of the transaction scheduler component within the STM, an issue we plan to cope with in future work. Additionally, we plan to investigate the viability of our modeling approach in Hardware Transactional Memory contexts.

References

[1] Richard M. Yoo and Hsien-Hsin S. Lee. Adaptive transaction scheduling for transactional memory systems. In *Proc. 20th Symposium on Parallelism in Algorithms and Architectures*, pages 169–178. ACM, 2008.

[2] Diego Rughetti, Pierangelo di Sanzo, Bruno Ciciani, and Francesco Quaglia. Machine learning-based self-adjusting concurrency in software transactional memory systems. In *Proc. 20th Int. Symposium on*

Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pages 278–285. IEEE, 2012.

[3] Aleksandar Dragojević and Rachid Guerraoui. Predicting the scalability of an stm a pragmatic approach. In *Proc. 5th ACM Workshop on Transactional Computing*. ACM, 2010.

[4] Pierangelo Di Sanzo, Francesco Del Re, Diego Rughetti, Bruno Ciciani, and Francesco Quaglia. Regulating concurrency in software transactional memory: An effective model-based approach. In *Proceedings of the Seventh IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2013.

[5] Diego Didona, Pascal Felber, Diego Harmanci, Paolo Romano, and Joerg Schenker. Identifying the optimal level of parallelism in transactional memory systems. In *Proc. International Conference on Networked Systems*, NETYS. Springer, 2013.

[6] Mohammad Ansari, Christos Kotselidis, Kim Jarvis, Mikel Luján, Chris Kirkham, and Ian Watson. Advanced concurrency control for transactional memory using transaction commit rate. In *Proc. 14th Int. Euro-Par Conference on Parallel Processing*, pages 719–728. Springer-Verlag, 2008.

[7] Aleksandar Dragojević, Rachid Guerraoui, Anmol V. Singh, and Vasu Singh. Preventing versus curing: avoiding conflicts in transactional memories. In *Proc. 28th ACM Symposium on Principles of Distributed Computing*, pages 7–16. ACM, 2009.

[8] Leonard Kleinrock. *Queueing Systems*, volume I: Theory. Wiley Interscience, 1975. (Published in Russian, 1979. Published in Japanese, 1979. Published in Hungarian, 1979. Published in Italian 1992.).

[9] Chi Cao Minh, JaeWoong Chung, Christos Kozyrakis, and Kunle Olukotun. STAMP: Stanford transactional applications for multi-processing. In *Proc. 4th IEEE Int. Symposium on Workload Characterization*, pages 35–46. IEEE, 2008.

[10] Pascal Felber, Christof Fetzer, and Torvald Riegel. Dynamic performance tuning of word-based software transactional memory. In *Proc. 13th ACM Symposium on Principles and Practice of Parallel Programming*, pages 237–246. ACM, 2008.

5 Appendix

We calculated p_k, in the case $x < k$, through the measured abort probability p_x according to the following approach. We note that, when there are x running transactions in the system, a transaction can conflict with any of the other $x-1$ running transactions. As a consequence, if p_a is the abort probability of a transaction when there is only another running transaction, the probability that no conflicts occur for a transaction when there are x running transactions (i.e. the probability that no conflicts occur with the other $x-1$ running transactions) is equal to $(1-p_a)^{x-1}$. Thus, the following equation holds:

$$p_x = 1 - (1-p_a)^{x-1}. \tag{7}$$

Solving by p_a the above equation, we have:

$$p_a = 1 - (1-p_x)^{\frac{1}{x-1}}. \tag{8}$$

Now, if we know the transaction abort probability p_x for a generic state x, we can calculate p_a using equation 8. Finally, we can calculate p_k for any $k \neq x$ (thus for any $k > x$) using k in place of x in equation 7.