Short Term Scientific Mission Final Report

Title: Performance Forecasting of Distributed Transactional Memory Systems
Author: Pierangelo Di Sanzo

1 Introduction

This report describes the work I carried out and the results that have been achieved in the context of my Short Term Scientific Mission (STSM) at INESC-ID, where I had the opportunity of closely collaborating with Prof. Paolo Romano and the researchers in his group. The STSM has been focused on the analysis and the development of performance forecasting tools for Distributed Transactional Memories (DTM) systems, which is one of main topic I deal with in my research work. Indeed, one of the main issues in Transactional Memory (TM) systems (both centralized and distributed) is the performance degradation that can be experienced when a sub-optimal system configuration is set. I remark that, unlike non-transactional systems, in TM-based applications performance is affected, in addition to contention on hardware resources (as CPU, system bus), also by contention on shared data. This increases the complexity of predicting the system's performance, because the various affecting factors are tightly intertwined and show complex inter-dependencies. The complexity is particularly exacerbated in the case of DTM, due to the presence of factors such as data distribution/replication and shared network links. As a consequence, developing performance analysis tools for a DTM system entails addressing a number of different aspects. In the literature, several approaches have been proposed for building performance computing system evaluation tools. These includes approaches based on simulation, analytical modeling (AM) and machine learning (ML), which are typically used as alternative techniques aimed to pursue the same goal: predicting the performance dynamics of computing systems when operating with different workload and configuration settings. As it is typical in the context of transactional system performance analysis studies, simulation aims at evaluating the system performance indicators by reproducing the dynamics of execution of transactions. AM, on the other hand, aims at quantitatively describing, via mathematical formulas, dependencies among system performance indicators and a number of parameters characterizing the system, as transaction arrival rate, transaction profiles and processing speed of operations. Finally, ML-based approaches "learn" statistical relations between performance indicators and other system parameters by observing only a subset of samples describing these relations.
Researcher of INESC-ID have recently developed TAS, (Transaction Auto Scaler) [1] a system for predicting the performance of DTM systems, which is used to guide (in an automatic fashion) the elastic scaling policies of the platform (in terms of number of nodes). TAS relies on a hybrid approach, leveraging both AM and ML. These approaches aim to capture complementary aspects affecting the system performance. More precisely, TAS exploits an analytical model for capturing the effects associated data contention (i.e. conflicts between transactions accessing shared data, which entail transaction aborts and restarts). Additionally, since aborting and restarting transactions entails further local hardware resource utilization, this is also predicted in TAS by the analytical model. On the other hand, exploiting such a white-box modeling approach for capturing the effect associated to network could be particularly hard. In

fact, especially for virtualized environments (as generally happens in the case of cloud infrastructures), very limited information could be known about the actual network infrastructure (e.g., network topology, shared links). Additionally, network delays also depend on the network throughput, which strongly depends on the transaction execution rate. Finally, the latter, in turn, affects the network throughput. All these factors make it very hard to use a white-box approach for network modeling. Thus, in TAS, a black-box approach, based on ML-techniques, has been used to predict the performance of the network primitives (in particular, the two-phase commit protocol [8]) used to synchronize the replicas of a DTM platform.
In my previous research work, I have contributed to the development of DAGS (DAta Grid Simulator) [2], a discrete event simulator for DTM systems incorporating models of various concurrency control and distributed data consistency algorithms. DAGS allows to predict a set of system performance indicators, as throughput, transaction response time and abort rate, while varying a number of system configuration parameters. As an example, configuration parameters include the number of clients, the number of nodes of the platform, the transaction profiles, the data object replication degree and the system model (open or closed).
DAGS exploits a white-box approach, where simulation clients and servers are implemented by means of objects, which incorporate algorithms that simulate the behavior of real clients and servers. These algorithms include, e.g., the transaction profile generator algorithm for the case of the clients, the local concurrency control algorithm and the distributed transaction commit algorithm for the case of the servers. All these algorithms can be easily tuned, through specific configuration parameters, or changed by the user. For what concerns the network model, DAGS offers the opportunity of using one or more simulation objects. E.g., a single object that generates network delays according to a given distribution (e.g. the exponential distribution) can be used. Alternatively, more simulation objects can be used for implementing more detailed simulation models, exploiting, e.g., different objects to simulate network switches, routers and links. Anyway, due to motivations remarked above in this report, using white-box models could not always be a feasible approach. Thus, the basic idea, which has been the main goal of my STSM, has been to enable DAGS to also provide an alternative modeling approach for the case of the network, i.e. a ML-based one, as in TAS. As a result of the work I carried out in my STSM, a new version of DAGS has been developed, where a machine learner can be queried for estimating the network delays associated with server to server communications. Specifically, when a server to server communication has to be simulated, DAGS passes to the machine learner a set of parameters defining the current system state (as the network throughput and the number of servers in the system), receiving as output the expected network delay. Then, DAGS uses this information to set the delivery time of the simulation event for the destination server in order to simulate the message sent through the network.
The tool I exploited for building the ML-based model of the network is Cubist [9], which has also been used in TAS. Cubist is a decision-tree regressor, providing the possibility of generating rule-based predictive models starting from sets of data. Specifically, Cubist can build decision trees choosing the branching attribute such that the resulting split maximizes the normalized information gain. For each leaf in the tree, Cubist places a multivariate linear model
In the rest of this report, details on the issues that have been addressed for developing the new version of DAGS are presented. Additionally, results of a preliminary validation study are provided and discussed. The report ends with a discussion about future directions about the development of DAGS.

2 System models and assumptions in TAS and DAGS

The machine learning-based network model developed by INESC-ID researchers is tailored for the TAS system model. System models and assumptions used in TAS and DAGS show some differences. These differences make them more or less suitable for different performance analysis scenarios. Additionally, they had an impact on the integration of DAGS with the ML-based model exploited in TAS. A comparison between system models and main assumptions in TAS and DAGS is provided in the follow.

In both TAS and DAGS the system is assumed to be composed by a number of homogeneous servers, each one with a fixed number of available threads processing transactions (however, both allow to easily extend the model for the case of heterogeneous platforms). The workload of the application is assumed to be characterized by different transaction profiles, which are mapped with different transaction classes. Anyway, in TAS, data access pattern of transactions is captured through the so-called Application Contention Factor (ACF). The ACF is a scalar metric, shown to be an invariant with respect to the system size/degree of parallelism of the system, which can be used to quantify, in a lightweight way, the degree of data contention generated by applications with arbitrary transactions' data access patterns. The ACF can be calculated by running the application and measuring the transaction execution rate, the transaction conflict probability and the average lock holding time (see [4] section 4.1.1). Conversely, DAGS does not use such an approach for characterizing the data contention of the application. DAGS allows to define the specific data access pattern of transactions by directly describing the specific sequence of accessed data objects for each transaction classes. (possibly, these sequences can be also expressed exploiting probability distributions). By using fine-grained simulation of the applications' data access patterns, DAGS allows to closely capture the dynamics even of the most complex/challenging workloads. This comes of course at the cost of having to develop simulation models of applications' data access pattern. On the other hand, TAS provides the advantage of not requiring a specific characterization of the data object access, as it is captured through the ACF. However, the definition of ACF is based on analytical assumptions (e.g. the poissonianity of the arrival rate of lock requests), whose validity may be challenged in complex workloads or at very high conflict rates. Essentially, this is one of the main difference that makes TAS and DAGS suitable for different performance analysis perspectives. Further, a current limitation of the system model of TAS is the assumption of full replication (i.e. a replica of each data object is stored on each server). Conversely, DAGS captures also the case of partial replication (i.e. a replica of each data object is stored on a sub-set of servers, according to an arbitrary placement).

As for hardware resource workload characterization, both TAS and DAGS require a workload profiling phase, where measurements of the service time required to execute a number of types of operations have to be performed. Particularly, TAS requires evaluating the average elapsed time between the transaction begin operation and the start of the prepare phase, the lock-contention probability, the lock request arrival rate and the average lock holding time. Additionally, the network throughput and the average CPU utilization of servers are required in TAS by the ML-based model. With respect to TAS, DAGS bases on a lower level workload characterization. It requires to evaluate the service times of different types of operations executed locally and remotely by transactions, including begin, read, write, prepare and commit operations. Additionally, the average service times of non-transactional code blocks executed between two consecutive transactions and between two consecutive operations of transactions are required.

For what concerns the network, the modeling approach of DAGS has already been described in the introduction of this report. As for TAS, the adopted ML-based model of the network provides as output the average round trip time (*rtt*) experienced by a server during the prepare phase, i.e. the elapsed time between sending the prepare message to other servers until receiving the last prepare reply massage by servers. The *rtt* is calculated as function of:

- the number of servers in the system,
- the number of thread per server executing transactions
- the distributed commit request rate,
- the average CPU utilization of servers, and
- the size of network messages for server to server communication.

Note that the CPU utilization of servers is also included in the network model because in the system model used in TAS the communication times between servers also include the CPU response time to prepare and send messages. The values of the above parameters are provided to the ML-based model by the analytical model. The latter, after having achieved the estimated value of the *rtt* by the ML-based model, recalculates the system performance indicators on the basis of this result. In this phase, also the values of the network throughput and the average CPU utilization of servers are recalculated by the analytical model. The recalculated values can be different with respect to those previously passed as input to the ML-based model. This is accounted by exploiting a method based on an iterative process. A graphical representation of this process is depicted in Figure 1, where p_n represents the value of a generic performance indicator at step *n*. At the first iteration step, input values that the model should receive by the network model are set to arbitrary values. The iteration process terminates when a given precision is reached by the estimation of the performance indicators, i.e. when the absolute differences between their values at step *n* and the step *n-1* are less then a given ε, which depends on the specific performance indicator.

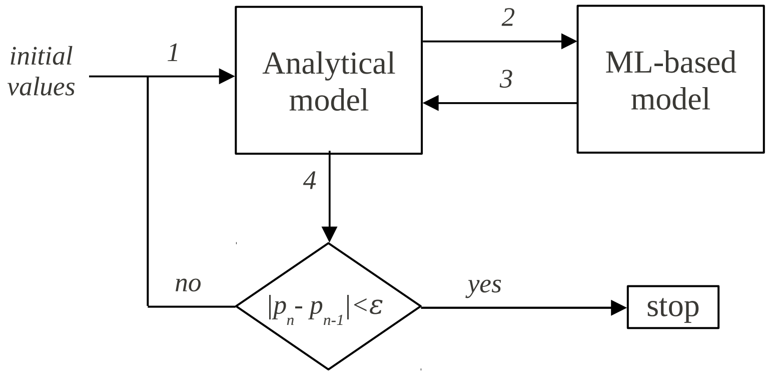


Figure 1: flow diagram of the iteration process. Numbers represent the sequence of actions executed for each iteration steps.

3 Integration of DAGS with the ML-based network model

In the literature, an iterative method, as that used for coupling TAS with the ML-based network model, has been used for the numerical resolution of various non-linear analytical models for transactional systems, as in [3,4,5,6]. In all these studies, it is shown that the iterative process converges when the workload is such that the utilization factor, for each resource in the system, also including data objects, is less then 1. Otherwise, the process never converges. It is known that demonstrating such a property is typically hard for non-trivial models. Indeed, in all the above mentioned studies, the convergence of the iteration processes has been only empirically shown. This has been done by comparing results achieved by executing a number of tests with different configurations of the real system and results achieved by the analytical models.

The first idea for integrating DAGS and the ML-based network model has been that of trying to exploit such a kind of approach. Thus, they have been coupled as depicted in Figure 1, where the DAGS simulation model has taken the place of the analytical model. However, as will be motivated in the follow, after a number of tests, it has been decided to evaluate a different interaction process. Hence, coupling the simulation model and the ML-based model entailed to deal with two main issues:

1) re-designing the interaction pattern between the DAGS simulation model and the ML-based model used in TAS, and
2) extracting the input data to be provided as input to the machine learning-based model from the simulator model.

The first point has been motivated by the fact that using an iterative process, exactly as it has been done in TAS, would require long execution time. In fact, such an iterative process, for each iteration step, entails: 1) executing a simulation run, 2) providing the output of the simulation to the ML-based model, 3) executing e new simulation run using the ML-based model output, and 4) comparing the output of the simulation run with the output of the previous iteration step. When executing a simulation run, an execution time is required to achieve statistical results for a given confidence interval. As it typically happens when dealing with simulations and analytical models, the difference between the simulation execution time of DAGS and the time required to solve the analytical model of TAS is non minimal. In some tests, the time to execute a simulation run in order to evaluate the system throughput with a 95% confidence interval and an amplitude less then 5% has been two order of magnitude greater then the time spent to solve the analytical model. To deal with this issue it has been tested another interaction pattern between DAGS and the ML-based model. This is described in the follow.

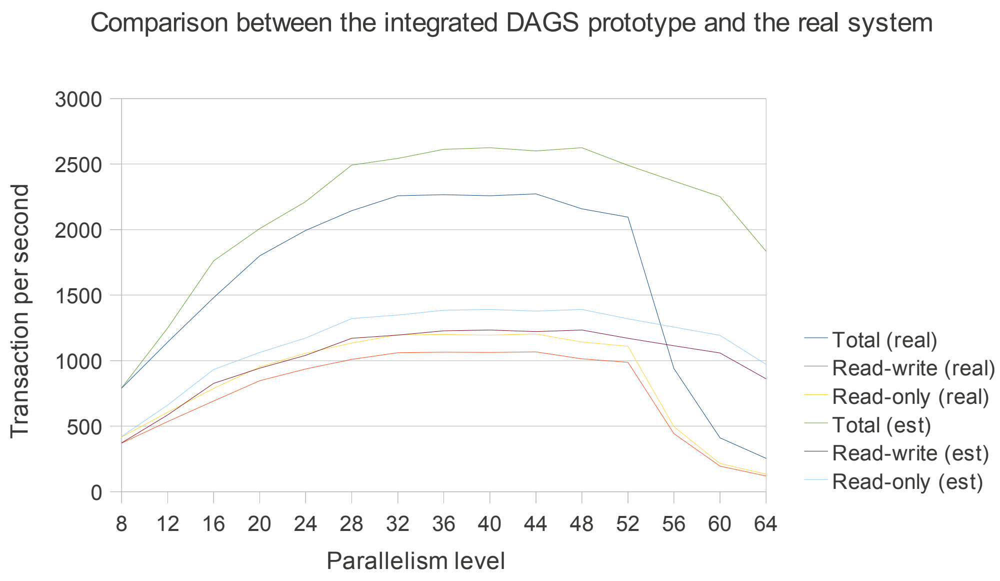
Re-designing the interaction pattern. The new interaction pattern that has been tested exploits only one simulation run executed by DAGS. During the simulation run, a number of queries to the ML-based model are performed. Input data to be provided to the ML-based model are calculated on-line during the simulation run, and a query is performed when the values of at least one input parameter of the ML-based model changes with respect the previously calculated value. Particularly, the input parameters whose values can change during a simulation run are the distributed commit request rate, the average CPU utilization of servers and the size of network messages for server to server communication. Note that non-minimal variations of the size of network messages mainly depend on the variation of the size of the prepare messages exchanged during the two-phase commit. The size of these messages depends on the size of the transaction write-set, that are directly accounted in DAGS for each execution of a transaction. Thus, at commit-time they are known in DAGS. Conversely, the values of the distributed commit request rate and the average CPU utilization of servers were not calculated in DAGS because they were unnecessary in the DAG simulation model. During a simulation run, these parameters, being time-dependent, have to be evaluated over a time windows. This has been addressed as described in the follow.

Extracting the input data for the ML-based model. Being the model used in DAGS a time-based simulation one, the values of the distributed commit request rate and the average CPU utilization of servers can be subject to fluctuations along the execution of the simulation run. As their values are used as input for the ML-based model, this could lead to fluctuations of the predicted values of the *rtt*. As it has been observed in a set of experiments, these fluctuations can strongly affect the evaluation of the system performance indicators, particularly the amplitude of the confidence intervals at the end of the simulations. In order to smooth the amplitude of fluctuations, it has been introduced in DAGS the exponential smoothing applied to the time series for the evaluation of the values of the distributed commit request rate and the average CPU utilization of servers. In this way, smoothing factors can be use in order to reduce the amplitudes, and, consequently, the simulation time to get a given confidence interval amplitude. Using this new interaction pattern, the time to execute a single simulation run has increased with respect to a single simulation run of the iterative solution. Overall, the measured reduction of total time need to terminate a test has been, on average, more than 80%.

4 Model validation

In this section, some preliminary evaluation results achieved with the version of DAGS integrated with the ML-based model are shown. Tests have been performed using Infinispan v4.0, a popular open-source NoSQL transactional data grid from JBoss/RedHat, and the TPC-C benchmark. In the used version of Infinispan, local concurrency control is based on the Two-Phase Locking (2PL) protocol (only write-locks are used) with eager lock acquisition, while distributed synchronization relies on Two-Phase Commit (2PC) protocol, where locks are acquired by a transaction on all data object replica.

Figure 2: Comparison between the integrated prototype and the real system



Some results are depicted in Figure 2 for a scenario with full data object replication. Transactions have been grouped in two classes, i.e. read-write transactions (43% of the total transactions) and read-only transactions (57%). The throughput for the two classes and the total throughput are shown in the figure with respect to the parallelism level in the system. The latter is given by the total number of concurrent threads executing transactions. The number of serves in the system was equal to 4, while the number of clients per server has been varied between 2 and 16.
These preliminary results show that the general shape of the curves achieved by the integrated prototype and the real system are similar, particularly concerning the saturation point (i.e. around a level o parallelism equal to 44). However, in some cases the relative prediction error of DAGS also reached the 20%. Possible motivations, which I'm currently addressing, include:

- the code instrumentation required to measure the cost of operations (as described in Section 3) entails the development of a number of additional probes that may introduce a non-negligible overhead in the systems. Thus, a reduction of the actual throughput can be expected. This overhead is not captured by DAGS, A detailed evaluation of the reduction of the throughput caused by the code instrumentation will clarify this aspect.

- during the profiling workload phase, the CPU service time of different operations showed differences till the 40% while varying the number threads in a server. While, in DAGS (as also in TAS) these are assumed to be independent by the number of threads. This may be due to various causes, particularly the presence of different shared cache levels (in all tests each server was equipped with eight hardware cores with level 2 and level 3 shared caches). These effect are, in general, hard to predict, unless exploiting cache performance models.

5 Future collaboration with the host institution and possible publication of the achieved results.

Currently, collaboration with INESC-ID researchers on topics addressed in my STSM is still in progress. Particularly, it is focusing on possible causes of the prediction error that has been measured in tests, as it has been discussed in Section 4. In the future, an extension of DAGS will be also explored. Specifically, given the detailed profiling phase required by DAGS to characterize the transaction access patters, the possibility to use in DAGS also the abstraction provided by the ACF will be evaluated. As a result, it is expected a noticeable reduction of the cost of the required profiling phase for cases where the transaction access pattern is particularly complex to analyze. Finally, an interesting research direction is related to the possibility of automatically synthetizing simulation models of the data access patterns of transactional applications based on static code analysis and statistical analysis based on patterns observed at run-rime.

Possible target conferences/workshops for publishing final results that will be achieved include, e.g., conferences focusing on simulation tools (as SIMUTools, where a paper on the original version of DAGS has already be published [2]), on performance modeling and evaluation tools (as ValueTools), on distributed transactional memories (as Euro-TM Workshop on Distributed Transactional Memory) .

6 Confirmation by the host institution of the successful execution of the STSM (by Prof. Paolo Romano)

Dr. Pierangelo Di Sanzo successfully pursued the objectives of the STSM, building what is, to the best of my knowledge, the first hybrid approach combining simulation models and machine-learning techniques to predict the performance of DTM systems.

The work performed during his STSM has paved the way to future research on a number of fronts. First, extending the preliminary evaluation study reported in this document. Second, refining the predictive models - both simulation and ML-based - to enhance their accuracy. Finally, automating the synthesis of simulation models for transactions' data access patterns.

Paolo Romano

Acknowledgments

Thanks to Paolo Romano for hosting me at INESC-ID and for supporting my work. A special thanks to Diego Didona for taking the time to work with me and for ready responses to all my questions.

References

[1] D. Didona, Paolo Romano, S. Peluso, F. Quaglia Transactional Auto Scaler: Elastic Scaling of In-Memory Transactional Data Grids The 9th International Conference on Autonomic Computing (ICAC 2012), San Jose, CA, USA, 17-21 Sept. 2011

[2] P. Di Sanzo, F. Antonacci, B. Ciciani, R. Palmieri, A. Pellegrini, S. Peluso, F. Quaglia, D. Rughetti and R. Vitali. A Framework for High Performance Simulation of Transactional Data Grid Platforms. In Proc. 16th International Conference on Simulation Tools and Techniques (SIMUTools2013).

[3] D. A. Menascé. 2002. Two-Level Iterative Queuing Modeling of Software Contention. In Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS '02). IEEE Computer Society, Washington, DC, USA

[4] Pierangelo Di Sanzo, Bruno Ciciani, Roberto Palmieri, Francesco Quaglia, and Paolo Romano. On the analytical modeling of concurrency control algorithms for software transactional memories: The case of commit-time-locking. Performance Evaluation, 69(5):187 – 205, 2012.

[5] Philip S. Yu, Daniel M. Dias, and Stephen S. Lavenberg. 1993. On the analytical modeling of database concurrency control. J. ACM 40, 4 (September 1993), 831-872.

[6] B. Ciciani, D. M. Dias, and P. S. Yu. 1990. Analysis of Replication in Distributed Database Systems. IEEE Trans. on Knowl. and Data Eng. 2, 2 (June 1990), 247-261.

[7] TPC Council. TPC-C Benchmark, Revision 5.11. Feb. 2010.

[8] Concurrencey Control and Recovery in Database Systems. P.A. Bernstein. Copyright 1987, Addison-Wesley.

[9] M. Kuhn, S. Witson, C. Keefer, and N. Coulter. Cubist Models for Regression. http://cran.r-project.org/web/packages/Cubist/vignettes/cubist.pdf, 2012. Last accessed: June 2013