Transactional Auto Scaler: Elasting scaling of NoSQL transactional data grids

Diego Didona, Paolo Romano
INESC-ID/IST, Lisbon, Portugal

Sebastiano Peluso, Francesco Quaglia
Sapienza University of Rome, Italy

***Abstract*—In this paper we introduce TAS (Transactional Auto Scaler), a system that relies on a novel hybrid analytical/machine-learning-based forecasting methodology in order to accurately predict the performance achievable by transactional applications executing on top of transactional in-memory data stores, in face of changes of the scale of the system. Applications of TAS range from on-line self-optimization of in-production applications, to the automatic generation of QoS/cost driven elastic scaling policies, and support for what-if analysis on the scalability of transactional applications.**

We demonstrate the accuracy and feasibility of TAS via an extensive experimental study based on a fully fledged prototypal implementation integrated with one of the most popular open-source transactional data grids (JBoss Infinispant©) and industry-standard benchmarks generating a breadth of heterogeneous workloads.

***Keywords*-Transactional data grids; Elastic Scaling; Analytical Performance model; Machine learning.**

I. Introduction

Context. The advent of commercial cloud computing platforms has led to the proliferation of a new generation of in-memory, transactional data platforms, often referred to as NoSQL data grids. This new breed of distributed transactional platforms (that includes products such as Red Hatt©'s Infinispan [1], Oraclet©'s Coherence [2] and Apache Cassandra [3]) is designed from the ground up to meet the elasticity requirements imposed by the pay-as-you-go cost model characterizing cloud infrastructures. By relying on a simpler data model (key-value vs relational), and employing efficient mechanisms to achieve data durability (in-memory replication vs disk-based logging) and to dynamically resize the cluster on top of which they are deployed [4], [5], these platforms allow non-expert users to provision a cluster of any size on the cloud within minutes.

This feature gives tremendous power to the average user, while placing a major burden on her shoulders. Previously, the same user would have had to work with system administrators and management personnel to get a cluster provisioned for her needs. However, removing the system administrator and the traditional capacity-planning process from the loop shifts the non-trivial responsibility of determining a good cluster configuration to the non-expert user [6].

Motivations. Unfortunately, forecasting the performance achievable by applications deployed on transactional data grids while varying the size of the underlying cluster is an extremely challenging task. The performance of transactional data platforms exhibit in fact strong non-linear behaviors as the number of nodes in the system grows, which are imputable to the simultaneous, and often inter-dependent, effects of contention affecting both physical (computational, memory, network) and logical (conflictings data accesses by concurrent transactions) resources.

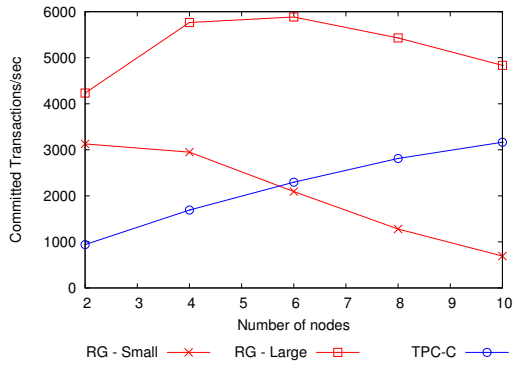
These effects are clearly shown in Figure 1, whose plots report results obtained by running two popular transactional benchmarking frameworks on top of the Infinispan data grid platform [1]: Radargun [1] and TPC-C [2]. Infinispan was configured to replicate data fully across the nodes of the data grid, and we selected benchmark configurations generating heterogeneous workloads in terms of: number of (read/write) operations executed within each transaction, percentage of read-only transactions, number of items in the whole dataset as well as size (in KB) of the individual objects manipulated by each operation.

A shown in Figure 1a, the scalability trends (in terms of the maximum throughput) of the three considered workloads are quite heterogeneous. The TPC-C benchmark scales almost linearly and the plots in Figure 1b and Figure 1c show that in this case the scalability is limited by a steady increase of contention at both the network and on the data level, which leads to a corresponding increase of the RTT and transaction abort probability.

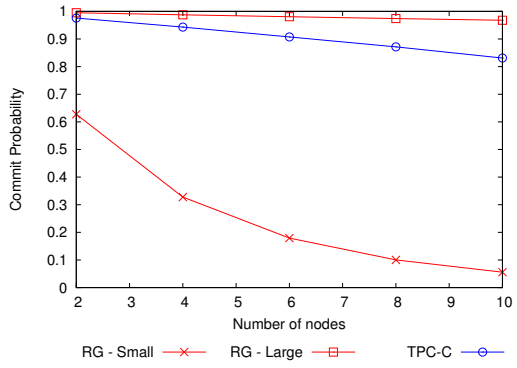
On the other hand the two Radargun workloads clearly demonstrate how the effects of high contention levels on logical and physical resources can lead to strongly non-linear scalability trends, even though, as in the case of accesses to a small dataset ("RG-Small"), the performance degradation of the network layer (in terms of RTT) is not so relevant.

Contributions. In this paper, we present Transactional Auto Scaler (TAS), a system that relies on a novel performance prediction methodology based on the joint usage of analytical and machine learning (statistical) models. The analytical models employed by TAS exploit the knowledge of the dynamics of the concurrency control/replication algorithm to forecast the effects of data contention using a white-box

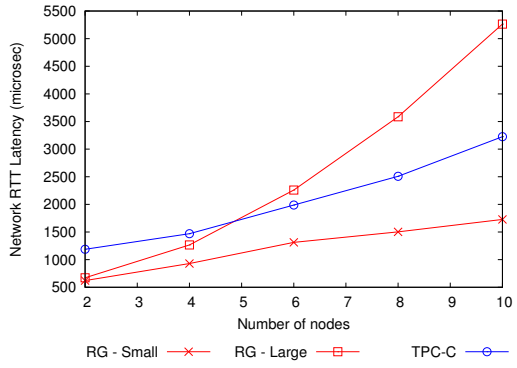
[1] http://sourceforge.net/apps/trac/radargun/wiki/WikiStart
[2] http://www.tpc.org/tpcc



(a) Throughput (Committed Transactions per second)



(b) Transaction Commit Probability



(c) Round-Trip Network Latency

Figure 1: Performance analysis of different data grids applications.

approach.

Statistical methods, on the other hand, are employed to forecast, using black-box machine-learning-based methods, the impact on performance due to shifts in the utilization of system level resources (e.g. CPU and network) caused by different replication schemes. This allows avoiding the explicit modelling of the interactions with system resources. This is not only a time-consuming and error-prone task given the complexity of current hardware architectures. It would also constrain the portability of our system (to a specific infrastructural instance), as well as its practical viability in virtualized Cloud environments where users have little or no knowledge of the underlying infrastructure.

While the methodology employed in TAS can be applied to a plethora of alternative replication/concurrency control mechanisms, one of the main contributions of this paper is the design of an innovative analytical performance model that targets the replication/concurrency control mechanisms used by Infinispan. One of the key innovative elements of the analytical performance model presented in this paper consists in the methodology introduced to characterize the probability distribution of accesses to data items. Existing white-box models of transactional systems [7], [8], [9], in fact, rely on strong approximations on the data accesses distribution,and require an a-priori time-consuming workload characterization phase to derive several parameters characterizing the data access distributions. In the presented model, conversely, we capture the dynamics of the application data access patterns via the abstraction of *Application Contention Factor* (ACF). ACF is a novel metric that exploits queuing theoretical arguments and a series of lock-related statistics (such as lock contention probability and average lock duration) measured (and dependant) on the current workload/system configuration, in order to derive a probabilistic model of the application's data access pattern that is independent from the current level of parallelism (e.g. number of concurrently active threads/nodes) and contention on physical resources (e.g. cpu or network).

We demonstrate the viability and high accuracy of the proposed solution via an extensive validation study based on industry standard benchmarks that generate a breadth of heterogeneous workloads for what concerns contention on both logical and physical resources. The results show that the overhead introduced by our monitoring and gathering of statistical information (in terms of reduction of the maximum achievable throughput) is negligible, and that the time required to solve the performance forecasting model is on the order of at most a few hundreds of milliseconds on commodity hardware.

The remainder of this paper is structured as follows. In Section II we discuss related research. The target data grid architecture of the TAS system is described in Section III. Section IV presents the forecasting methodology that we integrated in TAS, and Section V validates it via an extensive experimental study. Finally, Section VI concludes this paper.

II. Related Work

The present work is related to the literature on performance modeling and prediction for transactional systems. This includes both performance models for traditional database systems and related concurrency control mechanisms (see, e.g., [9], [10], [11]), approaches targeting more recent STM architectures (see, e.g., [12]), distributed/replicated transaction processing systems, such as [8], or multi-tier system [13]. With respect to these approaches, TAS presents two key differences: i) it relies on

analytical modelling only for capturing data contention dynamics, whereas it relies on black-block statistical methods to model the effects of contention on data resources; ii) from an analytical modelling perspective, in TAS we introduce a novel abstraction (ACF) that allows to concisely characterize and effectively reason about arbitrary transactional data access patterns.

Our work has clearly also relationships with systems that rely either on machine learning techniques or on analytical models in order to perform elastic scaling (or more, in general, performance forecasting) for other types of application domains (i.e. non-transactional applications), such as MapReduce [14], VM sizing [15], Grid resource brokering [16], [17] and online gaming [18]. Other solutions [19] rely on machine learning techniques to determine a policy for admission control of queries (read-only transactions) in centralized Database-as-a-Services. In addition to targeting different domains, we are not aware of solutions performing elastic scaling relying on the joint usage of analytical and machine learning based models.

Control theory techniques are also at the basis of several works in the area of self-tuning of application performance. These solutions often assume a linear performance model, which is possibly updated adaptively as the system moves from one operating point to another. For example, first-order autoregressive models are used to manage CPU allocation for Web servers [20]. Linear multi-input-multi-output (MIMO) models have been applied to manage different kind of resources in multi-tier application [21], as well as to allocate CPU resource for minimizing the interference between VMs deployed on the same physical node [22]. Compared to these adaptive linear models, the continuous non-linear models used by TAS to forecast both the logical and physical contention can accurately capture the system's entire behavior and allow optimized resource allocation over the entire operating space.

III. System Architecture

As already mentioned, TAS has been integrated into Infinispan [1], which is, at the time writing, one of the most popular open source in-memory NoSQL data grid on the market and the reference NoSQL data platform and clustering technology for the JBoss AS. Infinispan exposes a key-value store data model, and stores data entirely in-memory relying on replication as its primary mechanism to ensure fault-tolerance and data durability. Infinispan provides support for transactions and for two main operational modes: partial vs full data replication, depending on whether data are replicated on a subset or on the whole set of nodes in the data grid.

While the hybrid analytical/machine-learning based methodology presented in this paper is sufficiently generic to be extended also to cope with partial replication schemes, in this paper, we focus on the full replication mode of Infinispan. This replication mode is recommended for small/medium scale platforms (e.g. composed of up to 10/20 nodes) [23], [24], as, in these settings, the cost of the replica-wide synchronization phase required upon transaction commit is typically largely out-weighted by the guarantee to avoid expensive remote interactions to fetch data replicated at different nodes during transaction execution.

As many other recent NoSQL databases, Infinispan opts for sacrificing consistency in order to maximize performance: specifically, it does not ensure serializability [25] and only guarantees the Repeatable Read ANSI/ISO isolation level [26]. More in detail, Infinispan implements a lightweight, non-serializable variant of the multi-version concurrency control algorithm, which never blocks or aborts a transaction upon a read operation, and relies on an encounter-time locking strategy to detect write-write conflicts. Locks are first acquired locally during the transaction execution phase whenever a <key,value> pair is updated/inserted/deleted. Then, at commit time, a classic Two Phase Commit (2PC) protocol [25] is used to detect conflicts arisen with transactions concurrently executing on other replicas, as well as for guaranteeing transaction atomicity across the whole set of replicas. If the lock acquisition phase succeeds on all nodes, the transaction originator broadcasts back a commit message, in order to successfully apply the transacation's modifications on the remote nodes, and then it commits locally.

In presence of conflicting, concurrent transactions, however, the lock acquisition phase (taking place either during the local transaction execution or during the prepare phase) may fail due to the occurrence of (possibly distributed) deadlocks. Deadlocks are detected using a simple, user-tunable, timeout based approach. In this paper, we consider the scenario in which the timeout on deadlock detection is set to 0. This choice is due to the fact that, as the system scale grows, the distributed deadlock probability grows very quickly [27] and the default settings for the timeout value (10 seconds) impose a huge performance penalty that hampers significantly scalability. In fact, on the basis of an extensive manual tuning of this parameter, we could verify that, since in these in-memory systems the transaction turnaround time is typically on the order of a few milliseconds, throughput is maximized by setting the timeout value to 0 (at least in all the scenarios that we explored). This result is, indeed, not so surprising if one considers that analogous techniques are adopted in state of the art transactional memories [28].

As already mentioned, TAS relies on the periodic collection of system wide statistics on system's performance (e.g. transaction throughput and round-trip network latencies) and workload characterization (e.g. transactional arrival rate and number of writes per transaction) to instantiate its performance forecasting models. In order to maximize portability, rather than using additional monitoring frameworks to gather this information, we use a dedicated (i.e. not shared with the application) key-value container hosted by the same Infinispan instance to disseminate this data. At the end of each monitoring period (whose default value is set to 1 minute in our system) each replica stores the

statistic information that it locally collected into a dedicated entry of the key-value store. This rules out the possibility of conflicts, allowing to propagate updates outside of the boundaries of a transaction, further minimizing monitoring overhead. Given that the updates are extremely lightweight and infrequent, and given that Infispan's store used by the application is fully replicated, we opted to fully replicate also the key-value store used to monitor the data grid. This favours fault-tolerance, ensuring highly availability of monitoring information, and maximizing the flexibility of TAS deployments by allowing virtually each node of the data grid to instantiate, whether needed, the forecasting models described in the following. Note however that TAS can straightforwardly support architectures in which a dedicated node is used to instantiate the forecasting model (by simply including this node in the set of nodes joining the Infinispan instance used to gather monitoring information) and to drive the automatic elastic scaling of the system.

IV. Performance Forecasting Models

A. Analytical model

For space constraints we have to include the full description of the analytical model, and in particular details on the modelling of the lock contention dynamics, in appendix to this paper. In this section, however, we provide an overview of its main assumptions, goals and of probably represents one of its key innovative trait, namely the methodology introduced to characterize arbitrary transactional data access patterns.

Model Overview. Our analytical model uses mean-value analysis techniques to forecast the mean probability of transaction commit, the mean transaction duration as well as the maximum system throughput, supporting what-if analysis on parameters such as the degree of parallelism (number of nodes and possibly number of threads) in the system or shifts of workload characteristics (e.g. changes of transaction's data access patterns or of the percentage of read vs write transactions).

We treat the number of nodes in the system, and the number of threads processing transactions at each node as input parameters of the model. For the sake of simplicity, we will assume these nodes to be homogeneous in terms of computational power and available RAM (but extending the model to keep into account nodes with heterogeneous resources would be relatively straightforward).

We present the model assuming an open system in which transactions enter the system at a mean arrival rate λ_{Tx}; we denote with w the percentage of update transactions, and with N_l the number of write operations (which model, in general, operations that entail updating the state of the key-value store, e.g. put or remove operations) issued by a transaction before requesting to commit. We do not model explicitly the issuing of read operations as the concurrency control of Infinispan ensures that these are never blocked and can never induce an abort. We denote with T_{noCont} the time to execute a transaction since its beginning till the time in which it requests to commit, and assuming that it does not abort earlier due to lock contention; we denote with T_{prep} the mean time for the transaction coordinator to complete the first phase of 2PC, which includes broadcasting the prepare message, acquiring locks at all replicas, and gathering their replies.

Note that the value of T_{prep} (and, in principle, also of T_{noCont}) can vary significantly as the system scale changes, as an effect of the shift of the level of contention on physical resources (network in primis, but also CPU and memory). As these phenomena are captured in TAS using machine-learning techniques (described in Section IV-B), in the analytical model we will treat T_{prep} and T_{noCont} simply as input parameters.

Data Access Pattern Characterization. In order to compute the response time for a transaction, we need first to obtain the probability that it experiences a local or remote lock contention (that is, whether it requires a lock currently held by another transaction). Note that in the modelled concurrency control algorithm, lock contention leads to an abort of the transaction, hence the probability of lock contention, P_{lock}, and of transaction abort, P_a, coincide.

As in other analytical models of locking [11], [12], in order to derive the lock contention probability we model each data item as a server that receives locking requests at an average rate λ_{lock}, where each lock is held for an average time T_H. We can then approximate the probability of encountering lock contention upon issuing a write operation on a given data item with the utilization of the corresponding server (namely, the percentage of time the server is busy serving a lock request), which is computable as $U = \lambda_{lock} T_H$ [29] (assuming $\lambda_{lock} T_H < 1$).

The key innovative element of our analytical modelling approach is that it does not rely on any a priori knowledge about the probability of a write operation to insist on a specific datum. Existing techniques, in fact, assume uniformly distributed accesses on one [12] (or more [9]) set of data items of cardinality D, a-priori known, and compute the probability of lock contention on any of the data items simply as:

$$P_{lock} = \frac{1}{D} \lambda_{lock} T_H \tag{1}$$

The availability of information on D, and the assumption on the uniformity of the data access patterns strongly limits the employment of these models in complex applications, especially if these exhibit dynamic shifts in the data access distributions. Our model however is designed to be queried at run-time, while running already the user application in specific configuration settings for what concerns both workload characteristics and size of the underlying cluster. This means that we can exploit the availability of statistics information on P_{lock}, λ_{lock} and T_H in the current configuration to introduce a powerful abstraction that allows us to characterize the application data access pattern distribution

in a very concise, lightweight and pragmatical manner. Given P_{lock}, λ_{lock} and T_H, in fact, we can invert Eq. 1 and obtain what we call the Application Contention Factor (ACF) :

$$ACF = \frac{P_{lock}}{\lambda_{lock} T_H} \quad (2)$$

By equation 1, it is straightforward to see that $\frac{1}{ACF}$ can be interpreted as the size D of an "equivalent" set $\mathcal{DB}$ of data items such that:

1) the transactional business logic performs N_l write operations on disjoint data items selected uniformly from set $\mathcal{DB}$, and
2) the application would incur in the same contention probability that it experienced in the current configuration.

As we will show in Section V, despite its apparent simplicity, the abstraction of ACF is extremely powerful as it allows to characterize arbitrarily complex data access patterns (e.g. with strong skew or complex analytical representation) in terms of an easily tractable analytical model. This result represents the foundation on top of which we built an analytical model (reported in Appendix) of the lock contention dynamics, which allows to determine the contention probability that would be experienced by that same application in presence of different scenarios of workloads (captured by shifts of λ_{lock}), as well as of levels of contention on physical resources (that would lead to changes of the execution time of the various phases of the transaction life-cycle, capturable by shifts of the T_H).

B. Machine-learning-based modelling

TAS relies on black-box, machine-learning-based modelling techniques to forecast the impact on performance due to shifts of the level of contention of physical resources depending on workload's fluctuations or to re-sizing of the cluster on top of which the data grid is deployed. Developing white-box models capable of capturing accurately the effects on performance due to contention on hardware resources can in fact be very complex (or even unfeasible, especially in strongly virtualized cloud infrastructures) due the difficulty (or even impossibility) to gain access to detailed information on the exact dynamics of the many hardware components (processors, memory, network card) involved in the transaction processing.

In TAS we exploit the availability of a complementary white-box model of system's performance to formulate the machine-learning based forecasting problem in a way that differs significantly from traditional, pure black-box approaches to performance prediction. Conventional machine learning based techniques, e.g. [6], try to forecast some target performance metric p_2 in an unknown system configuration c_2, given the performance level p_1 and the demand of physical resources d_1 in the current configuration c_1. In TAS, instead, the analytical model can provide the machine learner with valuable estimates of the demand of physical resources d_2 in the target configuration c_2. Specifically, we use the analytical model to forecast what will be, in the target configuration c_2 the rate of transactions that will initiate a 2PC scheme (once reached their commit phase) as well as the percentage of CPU time consumed by the threads in charge of processing local transactions.

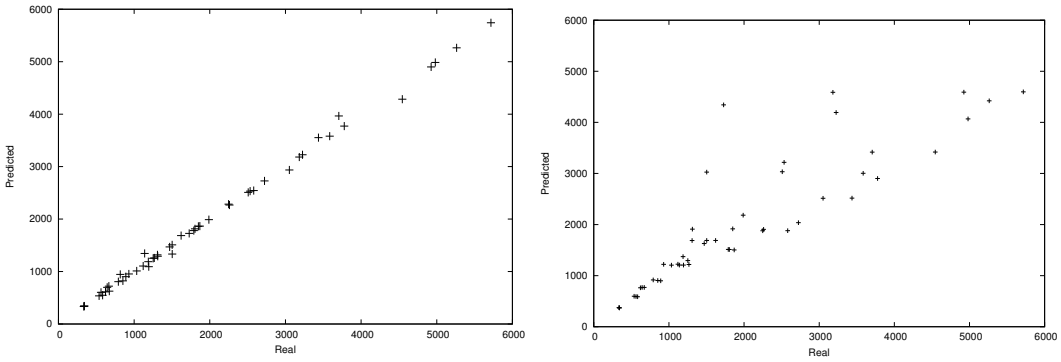
As already mentioned, contention on physical resources can have a direct impact on the execution time of two key phases of transactions' execution, namely the duration of the local transaction processing phase, denoted as T_{noCont}, and the network latency incurred in by transactions while executing the 2PC protocol, denoted as T_{prep}. We are here faced with a non-linear regression problem, in which we want to learn the value of continuous functions defined on multi-variate domains. We have experimented with two alternative machine learning techniques, namely decision-tree regressors and artificial neural networks.

As decision-tree regressor, we have used Cubist©. Analogously to classic decision tree based classifiers, such as C4.5 and ID3 [30], Cubist© builds decision trees choosing the branching attribute such that the resulting split maximizes the normalized information gain. However, unlike C4.5 and ID3, which contain an element in a finite discrete domain (i.e. the predicted class) as leaves of the decision tree, Cubist© places a multivariate linear model at each leaf.

For what concerns artificial neural networks, we have used radial basis function networks (RBFN), a special category of the feedforward neural networks architecture [31], which are known for their excellent performance as approximators of non-linear functions.

In order to build an initial knowledge base to train the machine learners, TAS relies on a suite of synthetic benchmarks to generate heterogeneous transactional workloads in terms of mean size of the messages generated by the replicas, memory footprint of the data grid, throughput (number of transactions activating 2PC per second) and execution duration of the transactional business logic. We run a short, but intensive initial training phase, in which we inject workload while varying the size of the cluster and the number of threads concurrently processing local transactions at each node. In our experiments we found that using a simple uniform sampling strategy allowed to achieve rather quickly (in less than an hour of automated testing) a satisfactory coverage of the parameter space, which is the reason why we did not decide to integrate more advance sampling mechanisms, such as adaptive sampling [32].

Once deployed on a data grid, the statistical gathering system of TAS periodically collects new samples of the workload and performance of the system. This allows for supporting periodic re-training of the machine learners and to incorporate in their knowledge base profiling data specific to the target user level applications.



(a) RBF Artificial Neural Network predictor.

(b) Regression Decision Tree predictor.

Figure 2: Accuracy of the machine-learning based T_{prep} predictions.

C. Coupling white-box and black-box modelling.

By the above discussion, the analytical and the statistical model are tightly intertwined: the analytical model relies on the predictions of the statistical model to obtain the values of the T_{prep} and T_{noCont} as input; the statistical model, on the other hand, uses as one of the input features of its model the transaction throughput forecast by the analytical model, thus obtaining an estimate on the level of resource contention in the target configuration.

Thus, we exploit a recursive scheme in which we initialize the analytical model with the currently measured values of T_{prep} and T_{noCont}, calculate the estimated throughput in the target configuration, and provide it as input feature to the machine learning to obtain a new value of T_{prep} and T_{noCont}. The process stops when the relative difference among the throughput output in two subsequent iterations falls beyond a given threshold. Proving the convergency of this fixed point iterative technique is out of the scope of this paper, but in all our experiments, this mechanism converged always after at most 10 iterations.

V. Validation

In this section we report the results of an experimental study aimed at evaluating the accuracy and viability of TAS when employed to forecast the performance of two well-known benchmarks for transactional systems, namely TPC-C and Radargun. The former is a standard benchmark for OLTP systems (of which we ported an implementation to execute on top of Infinispan), which portrays the activities of a wholesale supplier and generates mixes of read-only and update transactions with strongly skewed access patterns and heterogeneous duration. Radargun, instead, is a benchmarking framework specifically designed to test the performance of distributed, transactional key-value stores. The workloads generated by Radargun are much simpler and less diverse than TPC-C's ones, but have the advantage of being very easily tunable, thus allowing assessing the accuracy of TAS in a wide range of workload settings.

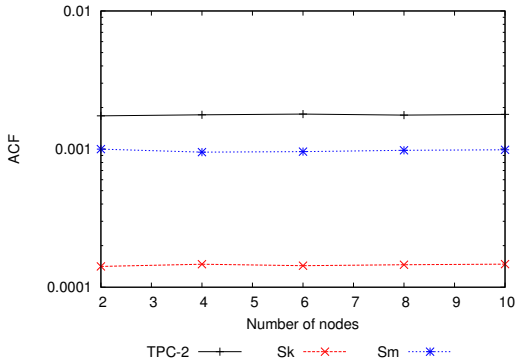
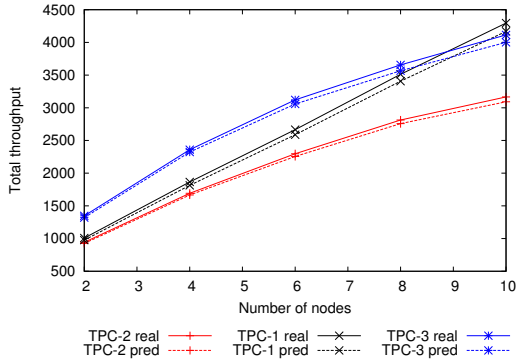


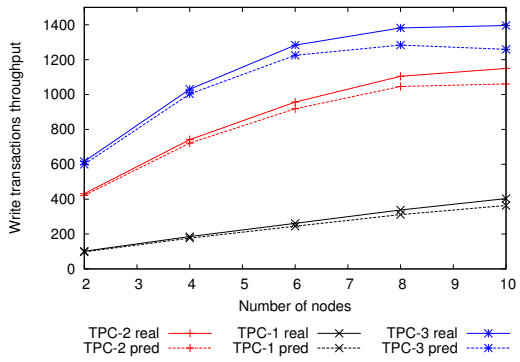
Figure 3: ACF of heterogeneous benchmarks.

For TPC-C we consider three different workload scenarios. The first, which we denote as TPC-1, is a read dominated workload (containing 90% read-only transactions) which generates reduced contention on both physical and data resources as the scale of the cluster grows. The second (TPC-2) and third (TPC-3) TPC-C workload, instead, include around 50% of update transactions and generate, respecitvely, moderate and high contention in particular at the data level.

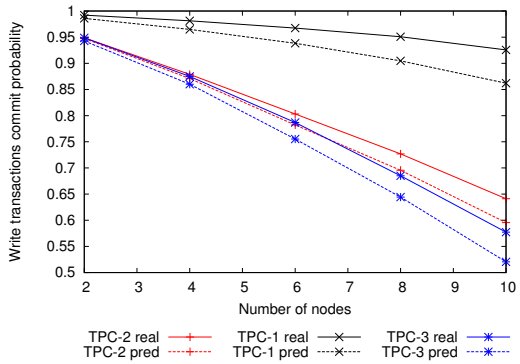
Also for Radargun we consider three workloads, but in this case we inject write-intensive traffic. This choice is motivated by the fact that update transactions, when compared to read-only transactions, generate a much higher contention level both on physical and logical resources. This makes several of the Radargun workloads much more complex to forecast than TPC-C's ones. The first workload (Sk) generates transactions that issue 10 writes distributed on a set of 100K keys and selected according to a highly skewed distribution (as defined by the NuRand(10000,8191), used by several TPC benchmarks). The second (La) and third (Sm) workload use a uniform data access pattern, with the former (La) performing a single put operation over a set of 100K data items, and Sm updating in each transactions 10 data items selected over a set of cardinality 1K.



(a) Throughput (read & write txs) for the TPC-C benchmark.

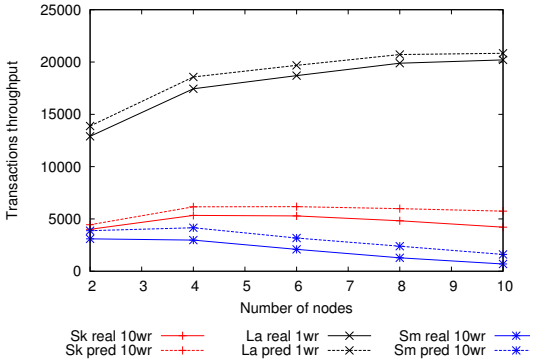


(b) Throughput (write txs only) for the TPC-C benchmark.

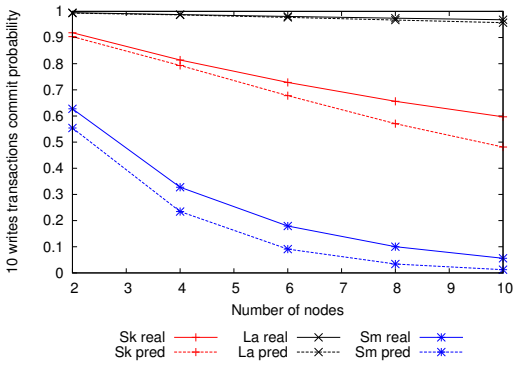


(c) Commit Probability (write txs only) for the TPC-C benchmark

Figure 4: Validation based on the TPC-C benchmark.



(a) Throughput (write txs only) for the Radargun benchmark.



(b) Commit Probability (write txs only) for the Radargun benchmark

Figure 5: Validation based on the Radargun benchmark.

All the results reported in this section were collected using a private cloud of 10 servers equipped with two 2.13 GHz Quad-Core Intel(R) Xeon(R) E5506 processors and 8 GB of RAM, running *Linux 2.6.32-33-server* and interconnected via a private Gigabit Ethernet.

We start by assessing the accuracy of the machine learners built using the synthetic benchmarking suite described in Section IV-B. To this end we use as test data set the whole set of aforementioned TPC-C and Radargun workloads. We focus on the forecasting of T_{prep}, since in all the explored settings we observed negligible shifts of the value of T_{noCont} in face of changes of the cluster size[3]. In order to evaluate the accuracy of the machine learning model in isolation (i.e. decoupling it from the analytical model), in this experiment we provide the machine learners with the correct guess of the target throughput. The scatter-plots reported in Figure 2 show that both the decision-tree regressor and the RBF networks attain a good prediction accuracy, even though the performance of RBF networks shine achieving an impressively high correlation factor (0.99 vs 0.81 for Cubist).

In Figure 3 we report the ACFs obtained when considering a subset of the considered workloads (this is done only for space constraints, as the ACF in the remaining scenarios confirm the presented results), which includes two strongly skewed workloads, namely TPC-2 and Sk, and a workload exhibiting uniform data access patterns, namely Sm. The plots show that, once fixed an application workload, and even when considering very skewed workloads, the ACF represents an invariant as the size of the underlying data grid varies. This confirms the appropriateness of the ACF to characterize application's data access patterns in a way

[3]This is due to the fact that in the considered settings, the system bottleneck is consistently the network rather than the CPU.

that is independent from the current degree of parallelism in the system (unlike for instance the transaction commit probability) and of the actual data access pattern distribution.

Let us now evaluate the accuracy of final performance prediction of TAS, in terms of maximum throughput achievable by the system as a function of the number of nodes in the data grid. For this experiment we coupled the analytical model with the machine learner based on RBF Artificial Neural Networks. We report in Figure 4 the forecasts of TAS for the considered TPC-C workloads, and in Figure 5 those for the Radargun workloads, contrasting them with the actual performance values attained by Infinispan. The experimental data clearly demonstrate the ability of TAS to predict with high accuracy non linear trends for what concerns not only transaction throughput, but also important intermediate statistics such as commit probability. More in detail, TAS achieves a remarkable average relative error (defined as $\frac{|real-pred|}{real}$) on the predicted throughput of 2%, with a maximum of 3.5%. As anticipated, the workloads of Radargun are more challenging, but also in this case the average relative error is on the order of 15%.

We conclude by remarking that, in all our experiments, the performance attained with or without the monitoring framework enabled were indistinguishable. Also, the time required to instantiate and solve an TAS query is on the order of a few hundreds of milliseconds, highlighting the practical viability of the proposed solution to support on-line what-if analysis and automatize elastic scaling.

VI. Conclusions

In this paper we introduced TAS (Transactional Auto Scaler), a system designed to accurately predict the performance achievable by applications executing on top of transactional in-memory data grids, in face of changes of the scale of the system. Applications of TAS range from on-line self-optimization of in-production applications, to the automatic generation of QoS/cost driven elastic scaling policies, and support for what-if analysis on the scalability of transactional applications.

TAS relies on a novel hybrid analytical/machine-learning based forecasting methodology that operate synergically according to a divide-and-conquer approach: availability of precise knowledge of the concurrency control scheme/replication procol is exploited to derive a white-box analytical model of data contention; black-block statistical techniques are instead used to capture the effect on contention on physical resources (CPU, memory, network) while avoiding explicit modelling of the interactions with system resources, which is not only complex and time consuming given the complexity of current hardware architectures, but is also normally unviable in virtualized Cloud environments where users have little or no knowledge of the underlying infrastructure.

We demonstrated the viability and high accuracy of the proposed solution via an extensive validation study based on industry standard benchmarks.

References

[1] Red Hat / JBoss, "JBoss Infinispan," http://www.jboss.org/infinispan, 2011.

[2] Oracle, "Oracle Coherence," http://www.oracle.com/technetwork/middleware/coherence/overview/index.html, 2011.

[3] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, 2010.

[4] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proc. STOC*, 1997.

[5] N. Raghavan and R. Vitenberg, "Balancing the communication load of state transfer in replicated systems," in *SRDS*, 2011.

[6] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11, 2011.

[7] P. di Sanzo, B. Ciciani, F. Quaglia, and P. Romano, "A performance model of multi-version concurrency control," in *MASCOTS*, 2008.

[8] B. Ciciani, D. M. Dias, and P. S. Yu, "Analysis of replication in distributed database systems," *IEEE Trans. Knowl. Data Eng.*, vol. 2, no. 2, 1990.

[9] Y. C. Tay, N. Goodman, and R. Suri, "Locking performance in centralized databases," *ACM Trans. Database Syst.*, vol. 10, 1985.

[10] D. A. Menascé and T. Nakanishi, "Performance evaluation of a two-phase commit based protocol for ddbs," in *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, ser. PODS '82, 1982.

[11] P. S. Yu, D. M. Dias, and S. S. Lavenberg, "On the analytical modeling of database concurrency control," *J. ACM*, vol. 40, 1993.

[12] P. D. Sanzo, B. Ciciani, F. Quaglia, and P. Romano, "Analytical modelling of commit-time-locking algorithms for software transactional memories," in *Proc. 35th International Computer Measurement Group Conference (CMG)*, 2010.

[13] J. Dejun, G. Pierre, and C.-H. Chi, "Resource provisioning of web applications in heterogeneous clouds," in *Proc. of USENIX conference on Web application development*, USENIX Association, 2011, pp. 5–5.

[14] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11, 2011.

[15] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. A. B. Fortes, "Fuzzy modeling based resource management for virtualized database systems," in *MASCOTS*, 2011.

[16] D. Bruneo, F. Longo, M. Scarpa, and A. Puliafito, "Performance analysis of job dissemination techniques in grid systems," *Concurr. Comput. : Pract. Exper.*, vol. 23, 2011.

[17] E. Elmroth and J. Tordsson, "A grid resource broker supporting advance reservations and benchmark-based resource selection,", *State-of-the-art in Scientific Computing*, 2006.

[18] V. Nae, A. Iosup, and R. Prodan, "Dynamic resource provisioning in massively multiplayer online games," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 380–395, March 2011.

[19] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. HacigümüŞ, "Activesla: a profit-oriented admission control framework for database-as-a-service providers," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011.

[20] Z. Wang, X. Zhu, and S. Singhal, "Utilization and slo-based control for dynamic sizing of resource partitions," in *DSOM*, 2005.

[21] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proc. of EuroSys*, 2009.

[22] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10, 2010.

[23] N. Schiper, P. Sutra, and F. Pedone, "P-store: Genuine partial replication in wide area networks," in *SRDS*, 2010.

[24] JBoss Infinispan, "Infinispan Cache Mode," https://docs.jboss.org/author/display/ISPN/Clustering+modes, 2011.

[25] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[26] 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*, ser. SIGMOD '95, 1995.

[27] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '96, 1996.

[28] D. Dice, O. Shalev, and N. Shavit, "Transactional locking ii," in *In Proc. of the 20th Intl. Symp. on Distributed Computing*, 2006.

[29] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[30] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[31] M. J. L. Orr, "Introduction to radial basis function networks," 1996.

[32] S. Thompson, *Sampling*, ser. Wiley series in probability and statistics. Wiley, 2002.

APPENDIX

ANALYTICAL MODEL

Before starting with the detailed description of the model, we illustrate the main assumptions it is based on and introduce the basic notations which be used throughout the discussion. We denote with ν the number of nodes in the data grid, and with θ the number of threads processing transactions locally originated at each node (or, equivalently, transactions that are dispatched by some load balancing component in charge of distributing load across the data grid); from now on, such transactions will be referred to as "local", whereas the ones which are originally served by another node will be referred to as "remote" .

Moreover we assume that write operations are uniformly distributed across the duration of the local execution of a transaction. Finally, we will assume that the system is stable: this means that all parameters are defined to be either long-run averages or steady-state quantities and transactions arrival rate does not exceeds service rate.

For the sake of presentation, we report in Table I the list of the key notations used while presenting the model.

Lock Contention Modelling

Let us now describe how we exploit the notion of ACF in the remainder of model to capture transactions' contention dynamics. Denoting with λ^l_{lock}, respectively λ^r_{lock}, the lock request rate generated by local, respectively remote transactions, on a given node, we can compute them as:

$$\begin{aligned} \lambda^l_{lock} &= \frac{\lambda_{Tx} \cdot w \cdot \tilde{N_l}}{\nu} \\ \lambda^r_{lock} &= \tilde{N_r} \cdot \lambda_{Tx} \cdot w \cdot \frac{\nu - 1}{\nu} \cdot P_p \end{aligned}$$

where we have denoted with P_p the probability for a transaction to reach the prepare phase (i.e. not aborting earlier), and with $\tilde{N_l}$, respectively $\tilde{N_r}$, the number of successfully acquired locks on average by local, respectively remote, transactions, independently from whether they abort or commit (which will be computed shortly).

When a transaction executes locally, it can experience lock contention (and therefore abort) both with other local transactions and remote ones.We can therefore compute the probability of abort during local transaction execution, P^l_a, as follow:

$$P^l_a = P^l_{lock} = (\lambda^l_{lock} + \lambda^r_{lock}) \cdot ACF \cdot T_H$$

The probability P^r_a for a transaction T to encounter contention upon any lock request issued during its prepare phase with a transaction T' on any node of the data grid can be instead approximated by considering exclusively the probability for T to contend with T' on the node $\nu_{T'}$ that generated the latter transaction. In fact, if T were to contend with T' at a node different from $\nu_{T'}$, then, with very high probability, T would also encounter lock contention with T' also when trying to complete its prepare phase on $\nu_{T'}$. As a consequence we can compute P^r_a as:

$$P^r_a = \lambda^l_{lock} \cdot ACF \cdot T^l_H$$

where T^l_H denotes the mean lock hold time held by a transaction on the node that originated it. Thanks to this approximation, we can consider as independent the remote abort probabilities for a transaction on different nodes.

By the above probabilities, we can compute the probability that i) a transaction reaches its prepare phase (P_p), ii) successfully completes its prepare phase on all the $N-1$ remote nodes (P_{coher}), and iii) commits (P_c):

$$\begin{aligned} P_p &= (1 - P^l_a)^{N_l} \\ P_{coher} &= (1 - P^r_a)^{N_l \cdot (\nu - 1)} \\ P_c &= P_p \cdot P_{coher} \end{aligned}$$

We can now compute the mean number of locks successfully acquired by a transaction, $\tilde{N_l}$, taking into account that it can abort during its execution:

$$\tilde{N_l} = P_p \cdot N_l + \sum_{i=1}^{N_l} P^l_a \cdot (1 - P^l_a)^{i-1} \cdot (i-1)$$

In order to compute $\tilde{N_r}$ we use a similar reasoning:

$$\tilde{N_r} = (1 - P^\dagger_a)^{N_l} \cdot N_l + \sum_{i=1}^{N_l} P^\dagger_a \cdot (1 - P^\dagger_a)^{i-1} \cdot (i-1)$$

with the exception that in this case we estimate the probability to incur in lock contention taking into account that there cannot be remote contention between two transactions originated by the same node:

$$P^\dagger_a = (\lambda^l_{lock} + \lambda^r_{lock} \cdot \frac{(\nu - 2)}{(\nu - 1)}) \cdot ACF \cdot T^\dagger_H$$

and $T^\dagger_H$ as:

$$T^\dagger_H = \frac{\lambda^l_{lock} \cdot T^l_H + \lambda^r_{lock} \cdot \frac{(\nu-2)}{(\nu-1)} \cdot T^r_H}{\lambda^l_{lock} + \lambda^r_{lock} \cdot \frac{(\nu-2)}{(\nu-1)}}$$

In order to compute the aforementioned probabilities, we need to obtain the mean holding time for a lock. To this end let us define as $G(i)$ the sum of the mean lock hold

Symbol	Meaning
ν	Number of nodes in the system
θ	Threads per node
λ_{Tx}	Transactions arrival rate
w	Percentage of write transactions
N_l	Average number data item updated by a write transaction
T_{noCont}	Successful local execution time of a transaction
T_{comm}	Time necessary to process the commit of a transaction
T_{roll}	Time necessary to perform a rollback
T_{prep}	Network round trip time
T_{Ro}	Execution time of a read-only transaction

Table I: Main notations used in the analytical model.

time over i consecutive lock requests (recalling that we are assuming that the average time between two lock requests is equal to $\frac{T_{noCont}}{N_l}$):

$$G(i) = \sum_{i=1}^{N_l} \frac{T_{noCont}}{N_l} \cdot i$$

We can then compute the local lock hold time as the weighted average of three different lock holding times, referring to the case that a transaction aborts locally (H_l^{la}), remotely (H_l^{ra}) or successfully completes (H_l^c).

$$T_H^l = H_l^{la} + H_l^{ra} + H_l^c$$

$$H_l^{la} = \sum_{i=2}^{N_l} P_a^l \cdot (1 - P_a^l)^{i-1} \cdot \frac{1}{i-1} \cdot G(i-1)$$

$$H_l^{ra} = P_p \cdot (1 - P_{Coher}) \cdot \frac{1}{N_l} \cdot [T_{prep} + G(N_l)]$$

$$H_l^c = P_p \cdot P_{Coher} \cdot \frac{1}{N_l} \cdot [T_{prep} + G(N_l)]$$

Let us now compute remote lock hold time, T_h^r. We neglect the lock holding times for transactions that abort while acquiring a lock on a remote node, as in this case locks are acquired consecutively (without executing any business logic between two lock requests). On the other hand, if a remote transaction succeeds in acquiring all its locks, then it holds them until it receives either a commit or an abort message from the coordinator. Therefore we compute T_h^r as:

$$T_h^r = (1 - P_a^\dagger)^{N_l} \cdot [T_{prep} + (1 - P_a^r)^{N_l \cdot (\nu-2)} \cdot T_{com}]$$

where $(1 - P_a^\dagger)^{N_l}$ represents the probability for a remote transaction T executing its prepare phase at node n to successfully acquire all the locks it requests on n, and $(1 - P_a^r)^{N_l} \cdot (\nu - 2)$ represents the probability for T to successfully acquire its remote locks on the remainder $\nu - 2$ nodes.

Given that an update transaction can terminate its execution (either aborting or committing) in three different phases, its mean residence time in the system (R^W) is the average among these cases:

$$R^W = T_c + T_a^l + T_a^r$$

where

$$T_c = P_c \cdot (T_{noCont} + T_{prep} + T_{comm})$$

$$T_a^l = \sum_{i=1}^{N_l} [T_{roll} + (\frac{T_{noCont}}{N_l} \cdot i)] \cdot P_a^l \cdot (1 - P_a^l)^{i-1}$$

$$T_a^r = P_p \cdot (1 - P_{coher}) \cdot (T_{noCont} + T_{prep})$$

Considering also read-only transaction, the average residence time of a transaction in the system is

$$R = w \cdot R^W + (1 - w) \cdot R^{RO}$$

Model resolution

From the description of the analytical model, it is clear that there is a mutual dependency between the abort probabilities and other parameters, like the mean hold time. Thus, we exploit a recursive scheme in which we first initialize these probabilities to zero, calculate the depending parameters and from them we obtain a new set of abort probabilities for the next iteration; the process stops when the relative difference among input and output probabilities falls beyond a given threshold. In order to avoid loops, we don't initialize the new set of probabilities with the exact output of the former iteration, but we perform a binary search in the bidimensional space $[0, 1] \times [0, 1]$, deciding the value for parameter of the i-th step depending on the outcome of $i - 1$-th step. For the model evaluation, we set the threshold to 0.001 and typically the model converges in 11 iterations.

Extension to the model

The model we presented can be easily extended to compute the maximum achievable throughput - and thus the highest sustainable load - of the system given a multiprogramming level of θ threads per node. Exploiting Little's law [cit] we give iteratively in input to the model a value of $\lambda = \frac{\theta}{R}$ obtained from the residence time for a transaction computed in the previous iteration. We use as starting value $\lambda = \frac{N \cdot \theta}{T_{prep} + T_{noCont}}$, which is clearly the upper-bound for the maximum throughput achievable by the system.

In addition it is also possible to extend the model to capture a retry-on-abort logic: the number of restart for a transaction can be obtained by computing the expected values of two geometric distributions with parameters (P_p) and P_{coher}, like in [10]; the residence time R' will be the total amount of time needed to a transaction to be successfully served.

ACKNOWLEDGMENT

This work has been partially supported by the project Cloud-TM (co-financed by the European Commission through the contract no. 257784), the FCT project ARISTOS (PTDC/EIA- EIA/102496/2008) and by FCT (INESC-ID multiannual funding) through the PIDDAC Program Funds.