Transactional Auto Scaler: Elastic Scaling of In-Memory Transactional Data Grids

Diego Didona, Paolo Romano
I.S.T./INESC-ID, Lisbon, Portugal

Sebastiano Peluso, Francesco Quaglia
Sapienza, Università di Roma, Italy

ABSTRACT

In this paper we introduce TAS (Transactional Auto Scaler), a system for automating elastic-scaling of in-memory transactional data grids, such as NoSQL data stores or Distributed Transactional Memories. Applications of TAS range from on-line self-optimization of in-production applications to automatic generation of QoS/cost driven elastic scaling policies, and support for what-if analysis on the scalability of transactional applications.

The key innovation at the core of TAS is a novel performance forecasting methodology that relies on the joint usage of analytical modeling and machine-learning. By exploiting these two, classically competing, methodologies in a synergic fashion, TAS achieves the best of the two worlds, namely high extrapolation power and good accuracy even when faced with complex workloads deployed over public cloud infrastructures.

We demonstrate the accuracy and feasibility of TAS via an extensive experimental study based on a fully fledged prototype implementation, integrated with a popular open-source transactional in-memory data store (Red Hat's Infinispan), and industry-standard benchmarks generating a breadth of heterogeneous workloads.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling techniques,Measurement techniques,Performance attributes*

Keywords

Analytical Models, Performance Evaluation, Autonomic Provisioning, Distributed Software Transactional Memory

1. 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 Hat's Infinispan, Oracle's Coherence and Apache Cassandra [19]) is designed from the ground up to meet the elasticity requirements imposed by the pay-as-you-go cost model at the basis of the cloud computing paradigm. By relying on a simple data model (key-value vs relational), employing efficient mechanisms to achieve data durability (in-memory replication vs disk-based logging) and dynamically resizing the cluster on top of which they are deployed, these platforms allow non-expert users to provision a cluster of virtually any size within minutes.This gives tremendous power to the average user, while placing a major burden on her shoulders. Removing the classic capacity planning process from the loop means in fact shifting the non-trivial responsibility of determining a good cluster configuration to the non-expert user [26].

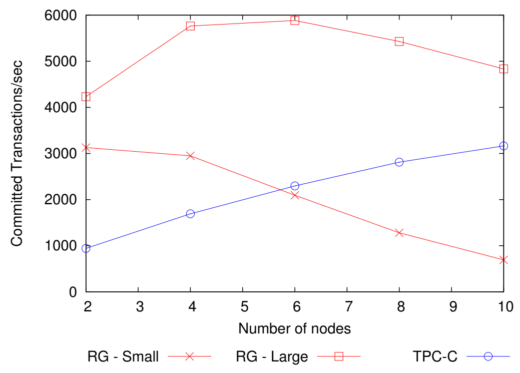
Motivations. Unfortunately, forecasting the scalability trends of real-life, complex applications deployed on distributed transactional platforms is an extremely challenging task. In fact, as the number of nodes in the system grows, the performance of these platforms exhibits strong non-linear behaviors. Such behaviors are imputable to the simultaneous, and often inter-dependent, effects of contention affecting both physical (computational, memory, network) and logical (conflicting data accesses by concurrent transactions) resources.

These effects are visible in Figure 1, which shows results obtained by running two transactional benchmarking frameworks on top of the Infinispan data grid platform [25]: Radargun[1] and TPC-C[2][35]. We deployed Infinispan over a private cluster encompassing a variable number of nodes and ran benchmarks generating heterogeneous workloads for what concerns the number of (read/write) operations executed within each transaction, the percentage of read-only transactions, the number of items in the whole dataset, as well as the size of the individual objects manipulated by each operation.

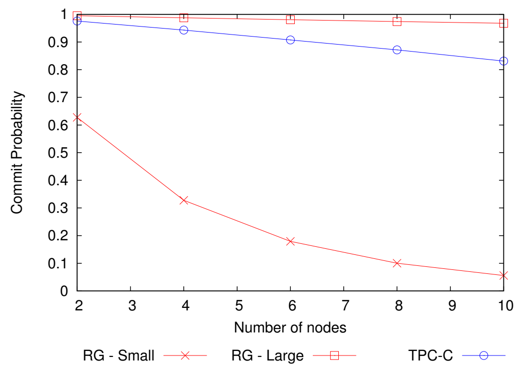
As shown in Figure 1a, the scalability trends (in terms of the maximum throughput) for the three considered workloads are quite heterogeneous. The TPC-C benchmark scales almost linearly and the plots in Figure 1b and Figure 1c show that scalability is hindered by a steady increase of contention at both network and data (i.e., lock) levels. This leads to a corresponding increase of the network round trip time (RTT) and of the 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.

Contributions. In this paper, we present Transactional Auto Scaler (TAS), a system that introduces a novel performance prediction methodology based on the joint usage of analytical and machine

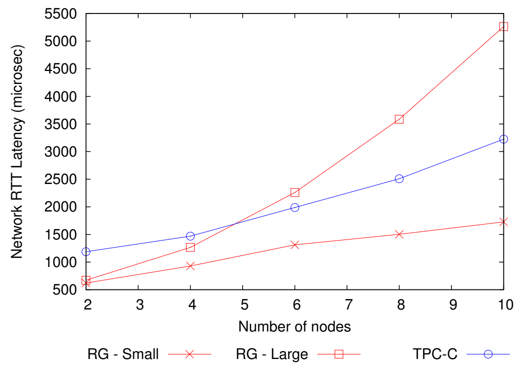
[1]http://sourceforge.net/apps/trac/radargun/wiki/WikiStart
[2]TPC-C is designed to operate on a relational database, hence we implemented a porting running directly on top of a key-value store such as Infinispan.



(a) Throughput (committed transactions per second)



(b) Transaction commit probability



(c) Round-trip network latency

Figure 1: Performance analysis of different data grid applications.

learning (statistical) models. The analytical model (AM) employed by TAS exploits knowledge of the dynamics of the concurrency control/replication algorithm to forecast the effects of data contention using a white-box approach. On the other hand, TAS exploits black-box, machine-learning (ML) methods to forecast the impact on performance due to shifts in the utilization of system level resources (e.g,. CPU and network) imputable to variations of the system's scale.

The synergic usage of AM and ML techniques allows TAS to take the best of these two, typically competing, worlds. On the one hand, the black-box nature of ML spares from the burden of explicitly modeling the interactions with system resources that would be otherwise needed using white-box, analytical models. 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.

On the other hand, analytical modeling allows to address two well known drawbacks of ML, namely its limited extrapolation power (i.e., the ability to predict scenarios that have not been previously observed) and lengthy training phase [5]. By exploiting *a priori* knowledge of the dynamics of data consistency mechanisms, AMs can achieve good forecasting accuracy even when operating in still unexplored regions of the parameters' space. Further, by narrowing the scope of the problem tackled via ML techniques, AM allows to reduce the dimensionality of the ML input features' space, leading to a consequent reduction of the training phase duration [5].

While the hybrid AM/ML methodology presented in this paper 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 in Infinispan. Similarly to other recent transactional data grids, e.g., [6, 19], Infinispan opts for guaranteeing a weaker consistency semantics than classic serializability.Specifically, Infinispan ensures Repeatable Read [2] by using an encounter time based write locking strategy and Two-Phase Commit.

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 transactions' access to data items. In fact, existing white-box models of transactional systems [8, 10, 28] rely on strong approximations on the data accesses distribution, e.g., uniformly distributed accesses on one or more sets of data items of fixed cardinality, which require complex and time-consuming workload characterization studies in order to derive the parameters characterizing the data access distributions. Conversely, in the presented model, we capture the dynamics of the application's data access patterns via a novel abstraction, which we call *Application Contention Factor* (ACF). ACF exploits queuing theory arguments and a series of lock-related statistics measured in (and dependent on) the current workload/system configuration, in order to derive, in a totally automatic fashion, a probabilistic model of the application's data access pattern that is independent of both the current level of parallelism (e.g., number of concurrently active threads/nodes) and the utilization of physical resources (e.g., CPU or network).

We demonstrate the viability and high accuracy of the proposed solution via a large scale evaluation study using both a private cluster and public cloud infrastructures (Amazon EC2), and relying on benchmarks that generate a breadth of heterogeneous workloads for what concerns contention on both logical and physical resources. The results also highlight that the overhead introduced by TAS' monitoring system 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 2 we discuss related research. The target data grid architecture of the TAS system is described in Section 3. Section 4 presents the forecasting methodology that we integrated in TAS, and Section 5 validates it via an extensive experimental study. Finally, Section 6 concludes this paper.

2. RELATED WORK

The present work is related to the literature on performance modeling and prediction for transactional systems. This includes performance models for traditional database systems and related concurrency control mechanisms (see, e.g., [28, 21, 34, 1]), approaches targeting more recent Software Transactional Memory architectures (see, e.g., [11]), and solutions dealing with distributed/replicated transaction processing systems, such as [8]. With respect to these approaches, TAS presents two key differences: i) it relies on analytical modeling only for capturing data contention dynamics, whereas it relies on black-box statistical methods to model the effects of contention on physical resources; ii) from an analytical modeling 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 also relationships with systems that rely solely on ML techniques to automate resource provisioning both in transactional [7, 15, 27, 33] and non-transactional application domains, such as MapReduce [17] and VM sizing [30]. As it will be shown in Section 5, the joint usage of AM and ML, which represents one of the key innovative characteristics of TAS, allows enhancing the extrapolation power and reducing the training time of pure ML-based performance predictors.

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 [31]. Linear multi-input-multi-output (MIMO) models have been applied to manage different kinds of resources in multi-tier applications [23], 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.

3. SYSTEM ARCHITECTURE

The architecture of TAS is depicted in Figure 2. Incoming transactions are dispatched by a front-end load-balancer towards the set of nodes composing the data grid. Periodically, statistics concerning load and resource utilization across the set of nodes in the data grid are gathered by a, so called, *aggregator* module.

Aggregated statistics are then fed to the *load predictor*, which serves the twofold purpose of forecasting future workload volumes and characteristics (e.g., ratio between read-only and update transactions, average number of read/write operations for read-only/update transactions), as well as detecting relevant workload shifts. The current TAS prototype relies on the Kalman filter algorithm [32] for load forecasting, and on the CUSUM [9] algorithm for distinguishing, in a robust manner, actual workload shifts from transient statistical fluctuations. Similar techniques have been employed in prior systems for automatic resource provisioning [7, 15], and have been shown to enhance the stability of the auto-scaling process.

The key innovative point of TAS, which represents the focus of this paper, is the methodology employed for predicting the performance of transactional applications when varying the number of nodes of the underlying data grid. More in detail, the *performance predictor* employed by TAS takes as input the workload characteristics (as output by the load predictor and/or aggregator) and the platform scale (i.e., the number of nodes to be used by the data grid), and outputs predictions on several key performance indicators (KPIs), including average response time, maximum sustainable throughput, transaction abort probability. As shown in Figure

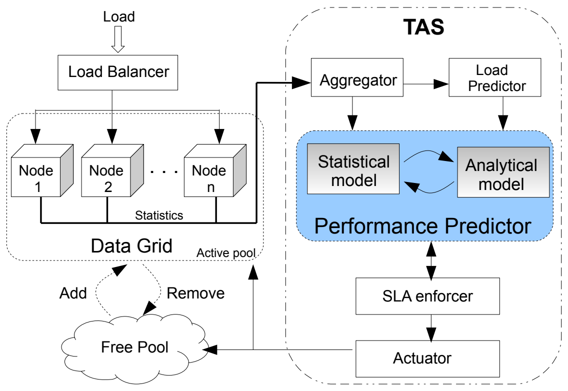


Figure 2: TAS reference architecture.

2, TAS relies on the joint usage of a white-box AM (to forecast the effects of data contention) and black-box ML techniques (to forecast the effects of contention on physical resources). A detailed description of the proposed performance forecasting methodology will be provided in Section 4.

The component in charge of querying the performance predictor is the *SLA enforcer*, which identifies the optimal platform configuration (in terms of number of nodes) on the basis of user-specified SLA and cost constraints. Our current prototype supports elastic-scaling policies that take into account constraints on cost, average response time and throughput. However, given that the performance predictor can forecast a number of additional KPIs (such as commit probability, or response time of read-only vs update transactions), our system lends itself to support more complex optimization policies involving constraints on additional performance metrics.

Finally, the *actuator* reconfigures the system by adding or removing (virtual) servers from the data grid. In order to maximize portability, TAS relies on δ-cloud[3], an abstraction layer which exposes a uniform API to automate provisioning of resources from heterogeneous IaaS providers (such as Amazon EC2, OpenNebula, RackSpace). The addition/removal of nodes needs of course to be coordinated also at the data grid level (not only at the IaaS level). To this end, TAS assumes the availability of APIs to request the join/departure of nodes from the data grid, which is a feature commonly supported by modern NoSQL data stores, such as Infinispan.

3.1 Infinispan Overview

As already mentioned, we selected as target platform for TAS a popular open source in-memory NoSQL data grid, namely Infinispan, which is developed by JBoss/Red Hat. At the time of writing, Infinispan is the reference NoSQL data platform and clustering technology for the JBoss AS, a mainstream open source J2EE application server. As TAS employs a white-box analytical model for capturing the effects of data contention on system's performance, in the following we provide an overview of the main mechanisms employed by Infinispan to ensure transactional consistency.

Infinispan exposes a key-value store data model, and maintains data entirely in-memory relying on replication as its primary mechanism to ensure fault-tolerance and data durability.

As other recent NoSQL platforms, Infinispan opts for weakening consistency in order to maximize performance. Specifically, it does not ensure serializability [3], but only guarantees the Repeatable Read ANSI/ISO isolation level [2]. More in detail, Infinispan implements a non-serializable variant of the multi-version concur-

[3] δCloud, http://http://deltacloud.apache.org/

rency 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. Write locks are first acquired locally during the transaction execution phase, which does not entail any interaction with remote nodes. At commit time, Two Phase Commit (2PC) [3] is executed. During the first phase (also called prepare phase), lock acquisition is attempted at all replicas, in order to detect conflicts with transactions concurrently executing on other nodes, as well as for guaranteeing transaction atomicity. If the lock acquisition phase is successful on all nodes, the transaction originator broadcasts a commit message, in order to apply the transaction'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, which is a typical approach for state of the art transactional memories [13] to achieve deadlock freedom. In fact, distributed deadlocks represent a major threat to system scalability, as highlighted by the seminal work in [16] and confirmed by our experimental results.

4. PERFORMANCE PREDICTOR

This section describes the performance prediction methodology employed by TAS. In Section 4.1, we introduce the analytical model used to capture data contention among transactions. Next, in Section 4.2, we present the machine learning based approach used to forecast the effects of contention on physical resources. Finally, in Section 4.3, we describe how to couple the two approaches.

4.1 Analytical Model

Our analytical model uses mean-value analysis techniques to forecast the probability of transaction commit, the mean transaction duration, and the maximum system throughput. This allows supporting what-if analysis on parameters like the degree of parallelism (number of nodes and possibly number of threads) in the system or shifts of workload characteristics, such as changes of the transactions' data access patterns.

The model treats the number of nodes in the system (denoted as ν) and the number of threads processing transactions at each node (denoted as θ) as input parameters. For the sake of simplicity, we will assume these nodes to be homogeneous in terms of computational power and available RAM, and distinguish only two classes of transactions, namely read-only vs update transactions. A discussion on how to extend the model and relax these assumptions will be provided in Section 4.1.4.

We denote with λ_{Tx} the mean arrival rate of transactions, and with w the percentage of update transactions, which perform, on average, a number N_l of write operations before requesting to commit. Note that, at this abstraction level, any operation that updates the state of the key-value store, e.g., put or remove operations, is considered a write operation. We say that a transaction is "local" to a node if it was activated on that node. Otherwise, we say that it is "remote".

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. However, we denote with $T_{localRO}$, resp. $T_{localWR}$, the average time to execute a read-only, resp. update, transaction, namely since its beginning till the time in which it requests to commit, assuming that it does not abort earlier due to lock contention (in case it is a write transaction).

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 $T_{localWR}$/$T_{localRO}$) can vary 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 via machine-learning techniques (described in Section 4.2), the analytical model treats T_{prep} and $T_{localWR}$/$T_{localRO}$ simply as input parameters.

Finally, we assume that the system is stable, with the meaning that i) all the parameters are defined to be either long-run averages or steady-state quantities and ii) the arrival rate of transactions does not exceed the service rate.

4.1.1 Data Access Pattern Characterization

In order to compute the response time for a transaction, we need first to obtain the probability that it experiences local or remote lock contention, that is whether it requires a lock currently held by another transaction. Note that in the modeled 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 AMs of locking [34, 11], 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}, and which takes an average time T_H before completing the "service of a lock request" (i.e., freeing the lock) . This level of abstraction allows to approximate the probability of experiencing 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 $U = \lambda_{lock}T_H$ [18] (assuming $\lambda_{lock}T_H < 1$).

The key innovative element of our AM 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 [11] (or more [28]) set(s) of data items of cardinality D (where D is assumed to be *a priori* known) and compute the probability of lock contention on any of the data items simply as:

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

Unfortunately, the assumption on the uniformity of the data access patterns strongly limit the employment of these models in complex applications, especially if these exhibit dynamic shifts in the data access distributions. We overcome these limitations by introducing a powerful abstraction that allows the on-line characterization of the application data access pattern in a lightweight and pragmatical manner. We call this abstraction *Application Contention Factor* (ACF) and define it as:

$$ACF = \frac{P_{lock}}{\lambda_{lock}T_H} \tag{2}$$

ACF has two attractive features that make it an ideal candidate to characterize the data access patterns of complex transactional applications:

1. It is computable on-line, on the basis of the values of P_{lock}, λ_{lock} and T_H measured in the current platform configuration. By Eq. 1, it is possible to see that $\frac{1}{ACF}$ can be alternatively interpreted as the size D of an "equivalent" dataset

accessed with uniform probability. Here, equivalent means that, if the application had generated a uniform access pattern over a dataset of size $D = \frac{1}{ACF}$, it would have incurred in the same contention probability experienced during its actual execution (in which it generated arbitrary, non-uniform access patterns).

2. As we will show in Section 5, even for applications with arbitrary, complex data access patterns (such as in TPC-C, whose access pattern is very hard to model analytically), ACF is an invariant with respect to the arrival rate, degree of concurrency in the system (i.e., number of nodes/threads generating transactions) and physical hardware infrastructure (e.g., private cluster vs public cloud platform).

The ACF abstraction represents the foundation on top of which we built the AM of the lock contention dynamics, to be discussed shortly. This model allows to predict the contention probability that would be experienced by an application in presence of different scenarios of workloads (captured by shifts of λ_{lock} or ACF), as well as of different levels of contention on physical resources (that would lead to changes of the execution time of the various phases of the transaction life-cycle, captured by shifts of T_H).

4.1.2 Lock Contention Model

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:

$$\lambda^l_{lock} = \frac{\lambda_{Tx} \cdot w \cdot \tilde{N}_l}{\nu}, \;\; \lambda^r_{lock} = \tilde{N}_r \cdot \lambda_{Tx} \cdot w \cdot \frac{\nu - 1}{\nu} \cdot P_p$$

where we have denoted with P_p the probability for a transaction to reach the prepare phase (i.e., not to abort earlier), and with $\tilde{N}_l$, respectively $\tilde{N}_r$, the number of locks successfully acquired on average by local, respectively remote, transactions, regardless of whether they abort or commit.

When a transaction executes locally, it can experience lock contention (and therefore abort) both with other local transactions and remote ones. By using Eq. 1, we can therefore compute the probability of abort during local transaction execution, P^l_a, as:

$$P^l_a = P^l_{lock} = (\lambda^l_{lock} + \lambda^r_{lock}) \cdot ACF \cdot T_H \qquad (3)$$

The probability P^r_a for a remote transaction T to experience 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 experience 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 for a local transaction. Thanks to this approximation, we can consider the remote abort probabilities for a transaction on different nodes as independent.

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:

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

Where we denoted with $T^{\dagger}_H$ the average lock holding time of the transactions with which it is possible to experience contention during the prepare phase, which we estimate 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 lock hold time over i consecutive lock requests (recalling that we are assuming that the average time between two lock requests is equal to $\frac{T_{localWR}}{N_l}$):

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

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^{la}_l), remotely (H^{ra}_l) or successfully completes (H^c_l):

$$\begin{aligned} T^l_H &= H^{la}_l + H^{ra}_l + H^c_l \\ H^{la}_l &= \sum_{i=2}^{N_l} P^l_a \cdot (1 - P^l_a)^{i-1} \cdot \frac{G(i-1)}{i-1} \\ H^{ra}_l &= P_p \cdot (1 - P_{Coher}) \cdot [T_{prep} + \frac{G(N_l)}{N_l}] \\ H^c_l &= P_p \cdot P_{Coher} \cdot [T_{prep} + \frac{G(N_l)}{N_l}] \end{aligned}$$

Let us now compute the remote lock hold time, T^r_h. 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^r_h as:

$$T^r_h = (1 - P^{\dagger}_a)^{N_l} \cdot [T_{prep} + (1 - P^r_a)^{N_l \cdot (\nu - 2)} \cdot T_{com}]$$

where $(1 - P^{\dagger}_a)^{N_l}$ 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 $(1 - P^r_a)^{N_l \cdot (\nu - 2)}$ represents the probability for T to successfully acquire its remote locks on the remaining $\nu - 2$ nodes.

Given that an update transaction can terminate its execution (either aborting or committing) in three different phases, its mean service time, denoted as T^W, can be expressed as:

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

where

$$T_c = P_c \cdot (T_{localWR} + T_{prep} + T_{comm})$$
$$T_a^l = \sum_{i=1}^{N_l} [T_{roll} + (\frac{T_{localWR}}{N_l} \cdot i)] \cdot P_a^l \cdot (1 - P_a^l)^{i-1}$$
$$T_a^r = P_p \cdot (1 - P_{coher}) \cdot (T_{localWR} + T_{prep})$$

Considering also read-only transaction, the average service time of a transaction, denoted as T, is:

$$T = w \cdot T^W + (1 - w) \cdot T^{localRO} \quad (4)$$

4.1.3 AM Resolution and Predicted KPIs

As in previous analytical models of transactional data contention [34, 12], also our model exhibits a mutual dependency between the abort probabilities and other parameters, such as the mean hold time. Prior art copes with this issue by using an iterative scheme in which abort probabilities are first initialized to zero. Next, the depending parameters are computed, and, on the basis of their values, a new set of abort probabilities is obtained and used in the next iteration; the process continues till the relative difference between the abort probabilities at two subsequent iterations becomes smaller than a given threshold.

It is known [34] that this iterative solution technique can suffer from convergence problems at high contention rates. We tackle this issue by adopting a binary search in the bi-dimensional space $[0, 1] \times [0, 1]$ associated with the abort probabilities (local and remote), which is guaranteed to converge at a desired precision $\epsilon \in (0, 1]$ after a number of steps $n \leq 1 + \lceil -log_2\epsilon \rceil$. This analysis was confirmed by our evaluation study, reported in Section 5, for which we set $\epsilon = 0.001$ and observed convergence in at most 11 iterations.

Once obtained the commit probability and the average transaction service time, the model can be employed to compute additional KPIs typically employed in SLA definition, such as maximum system throughput or percentiles on response times.

The maximum throughput can be computed by exploiting Little's law [20] in an iterative fashion. At the first step of the iteration, an upper-bound on system throughput is provided as input to the model, which is computed by assuming no conflicts and that all threads in the system constantly execute transactions. This corresponds to setting:

$$\lambda = \frac{\nu\theta}{w \cdot (T_{localWR} + T_{prep} + T_{comm}) + (1 - w) \cdot T_{localRO}}$$

At each step, a new value of λ is fed in input to the model, replacing the denominator of the above equation with the value of T (see Eq. 4) computed in the previous iteration, till convergence to the desired precision is reached.

In order to compute response time percentiles, it is possible to model each data grid node as a G/G/θ queuing system, i.e., a queue with θ servers subjected to arbitrary service and arrival rate distributions. One can then exploit the Köllerström's approximation [4] for the waiting time distribution of G/G/θ queues in the heavy-traffic case, namely when the queue utilization $\rho \simeq 1$. This result states that the approximate distribution of the waiting time, w, of a G/G/θ queue in heavy traffic is exponential and is given by:

$$P(w \leq t) \simeq 1 - e^{-\frac{2(1/\lambda - T_s)}{\sigma_u^2 + \frac{\sigma_v^2}{\theta^2}} t}$$

where σ_u is the inter-arrival time variance, σ_b^2 is the service time variance (both measurable at run-time, as done in other systems for automated resource provisioning, such as [27]), λ is the request arrival rate, and T_s is the average service time. The above formula can hence be used to compute the maximum arrival rate λ such that the response time is less than a given threshold y with probability k.

4.1.4 Extensions of the AM

The presented model lends itself to be extended in several directions. In the following we briefly overview some of the most interesting possible extensions.

Mix-aware modeling: extending our approach to account for multiple transaction classes having different characteristics (for instance in terms of data access pattern or duration of local execution) would require two main steps.

1. Extracting a characterization of the different transactional classes, including per-class information on ACF, abort probability, mean number of locks requested per transaction and local execution time, and of the ratio of each class in the mix. Identification of different transactional classes can be performed in a transparent way using classic clustering techniques, such as the one used, e.g., in [15].

2. Specializing the analytical model to forecast the contention probability (and depending statistics, such as throughput) per transaction-class. This result can be achieved in a relatively simple way by employing a methodology, similar to the one proposed in [34], in which the transaction conflict probability is computed taking into account the data access patterns (in our case captured by the ACF) of each single transaction class.

Heterogeneous platforms: as in prior approaches for automated resource provisioning for the Cloud [27, 26], heterogeneous platforms can be handled by using simple multiplication factors between servers depending on their hardware characteristics. For example, Amazon EC2 offers various instances (small, medium, large, etc.), each equipped with different hardware resources. Via a preliminary benchmarking study (using synthetic workloads, as in [26], or, whether possible, directly the target application, as in [27]), it is easy to determine scaling factors relating the performance achieved when deploying the application on different type of instances. For example, a medium instance performs 1.5 times better than a small instance, or a large instance provides 2x the throughput of a small instance. These scaling factors can then be applied to forecast the values of the parameters associated with the duration of local transaction execution, namely $T_{localRO}$ and $T_{localWR}$, when deploying the application on a different instance type.

4.2 Machine-Learning-Based Modeling

TAS relies on black-box, machine-learning-based modeling techniques to forecast the impact on performance due to shifts of the level of contention on physical resources depending on workload's fluctuations or to the re-sizing of the data grid. Developing white-box models capable of capturing accurately the effects on performance due to contention on hardware resources can in fact be

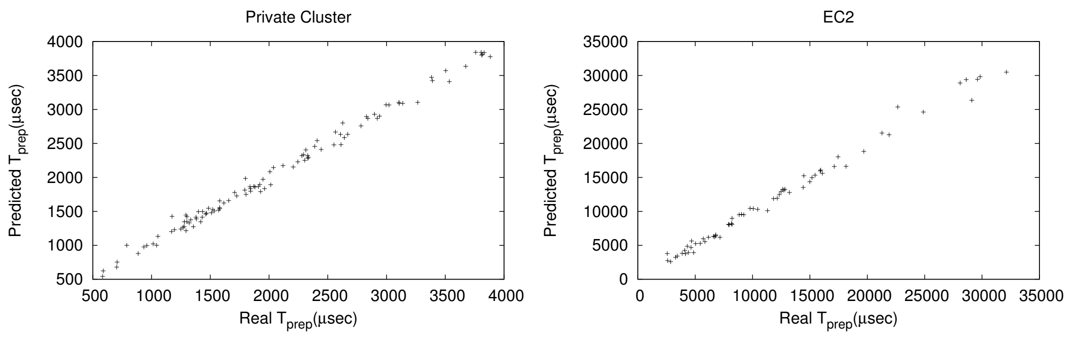


Figure 3: Accuracy of the machine-learning based T_{prep} predictions on the private cluster (left) and on EC2 (right).

very complex (or even non-feasible, especially in virtualized cloud infrastructures), given the difficulty to gain access to detailed information on the exact dynamics of hardware-level components.

In TAS we exploit the availability of a complementary white-box model to formulate the machine-learning based forecasting problem in a way that differs significantly from traditional, pure black-box approaches. Conventional machine learning based techniques, e.g., [26], try to forecast some 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 $T_{localWR}$ and $T_{localRO}$, 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 multivariate domains. Given the nature of the problem, we used, as machine learner, Cubist, a decision-tree regressor that approximates non-linear multivariate functions by means of piece-wise linear approximations. Analogously to classic decision tree based classifiers, such as C4.5 and ID3 [24], 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 elements 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.

In order to build an initial knowledge base to train the machine learner, TAS relies on a suite of synthetic benchmarks that generate heterogeneous transactional workloads in terms of mean size of messages, memory footprint at each node and network load (number of transactions that activate 2PC per second). Additional details on the criteria used to perform feature selection are provided, for space constraints, in [14]. During the initial, off-line, training phase, the benchmark suite injects 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 about one hour) a satisfactory coverage of the parameters' space, which is the reason why we did not decide to integrate more advanced sampling mechanisms, like adaptive sampling [29].

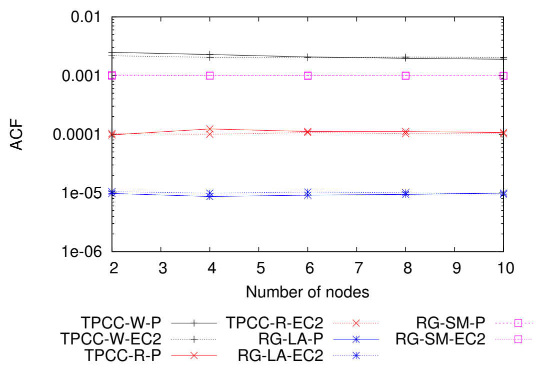


Figure 4: ACF for heterogeneous benchmarks.

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 to support periodic re-training of the machine learner and to incorporate in its knowledge base profiling data specific to the target user level applications.

4.3 AM and ML Coupling

By the above discussion, it is clear that the AM and the ML are tightly intertwined: the AM relies on the predictions of the ML to obtain the values of T_{prep} and $T_{localWR}$/$T_{localRO}$ as input; the ML, on the other hand, uses as one of the input features of its model the transaction throughput forecast by the AM, which represents an estimate on the level of resource contention in the target configuration.

For simplicity, the current prototype solves this problem by using the following fixed point iterative solution, which, in our experiments, has never shown convergence problems: the AM is initialized with the current values of T_{prep} and $T_{localWR}$/$T_{localRO}$, it outputs the estimated throughput in the target configuration, and provides it as input feature to the ML to obtain a new value of T_{prep} and $T_{localWR}$/$T_{localRO}$. The process is repeated till the requested precision is reached. Note that, also in this case, we may have employed a binary search technique analogous to the one described in Section 4.1.3. Such a technique provides stronger convergence properties, at the cost of a higher complexity since, in this scenario, it would need to operate on a three-dimensional space.

5. VALIDATION

In this section we report the results of an experimental study

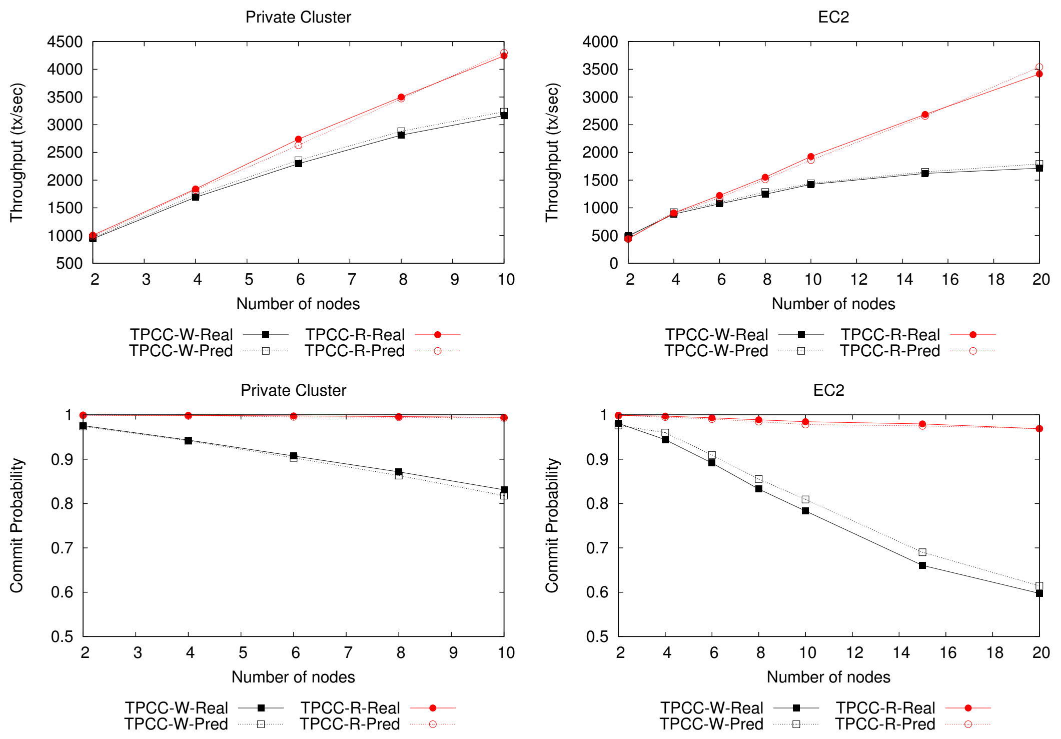


Figure 5: Validation using the TPC-C benchmark.

aimed at evaluating the accuracy and viability of TAS. Before presenting the results, we describe the workloads and experimental platforms used in our study.

Workloads. We consider two well-known benchmarks, already mentioned in Section 1, namely TPC-C and Radargun. The former is a standard benchmark for OLTP systems, which portrays the activities of a wholesale supplier and generates mixes of read-only and update transactions with strongly skewed access patterns and heterogeneous durations. Radargun, instead, is a benchmarking framework specifically designed to test the performance of distributed, transactional key-value stores. The workloads generated by Radargun are 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 wider range of workload settings.

For TPC-C we consider two different workload scenarios. The first, which we denote as TPCC-R, is a read dominated workload (containing 90% read-only transactions) that generates reduced contention on both physical and data resources as the scale of the cluster grows. The second (TPCC-W) includes around 50% of update transactions and generates a high data contention level.

For Radargun we also consider two workloads, denoted as RG-LA and RG-SM. Both workloads generate uniform data access patterns, but RG-LA performs, in each transaction, a single put operation over a set of 100K data items, yielding a very low contention rate. RG-SM, instead, updates in each transaction 10 data items selected over a set of cardinality 1K, thus generating a very high contention probability. We decided to use the Radargun workloads in our evaluation study because their data access patterns are particularly simple and easily predictable, thus allowing us to validate the correctness and semantics of the ACF abstraction.

Experimental Platforms. We use, as experimental test-beds for this study, both a private cluster and Amazon EC2. The private cluster is composed of 10 servers equipped with two 2.13 GHz Quad-Core Intel(R) Xeon(R) processors and 8 GB of RAM and interconnected via a private Gigabit Ethernet. For EC2 we used up to 20 Extra Large Instances, which are equipped with 15GB of RAM and 4 virtual cores with 2 EC2 Compute Units each.

ML validation. We start by assessing the accuracy of the machine learners built using the synthetic benchmarking suite described in Section 4.2. We focus on the forecasting of T_{prep}, since in all the explored settings we observed negligible shifts of the value of $T_{localRO}/T_{localWR}$ in face of changes of the cluster size. This is due to the fact that, in the considered settings, the system bottleneck is consistently the network rather than the CPU.

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 in Figure 3 report the results of 10-fold cross validation, highlighting that, on both the private cluster and on EC2, the ML attains a high prediction accuracy. Specifically, the correlation factor was around 99% in both cases, with an average absolute error equal to 500 micro-seconds for EC2 and around 60 micro-seconds for the private cluster. Note that, in practice, the relative error is similar on both platforms, since, on EC2, the maximum value of T_{prep} is around 10 times greater than the maximum T_{prep} value on the private cluster.

ACF validation. In Figure 4 we report the ACFs obtained when running both the TPC-C and Radargun workloads on EC2 and on the private cluster (note that we tag the curves obtained on the private cluster with the suffix "-P"). The plots confirm our finding, namely that, once fixed an application workload, the ACF represents an invariant across platforms of different scale, even when deployed on infrastructures of different nature (private vs public). It is noteworthy to highlight that the ACF value is equal to 1E-5,

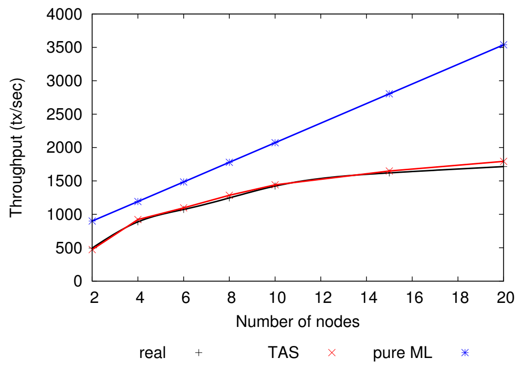


Figure 6: Comparing TAS with a pure ML approach.

resp. 1E-3, for the workloads RG-LA, resp. RG-SM. We recall that these workloads generate uniform accesses to datasets of size 100K, resp. 1K, items. Therefore, these results confirm that ACF can be interpreted as the inverse of the size of an equivalent, uniformly accessed, dataset.

AM/ML validation. Let us now evaluate the accuracy of the final performance predictions output by TAS when jointly using the AM and the ML. We use as KPIs the maximum throughput and commit probability. We report in Figure 5 the forecasts for the TPC-C workloads. For space constraints, we cannot include the plots for Radargun, however they show analogous trends. The experimental data demonstrate the ability of TAS to predict with high accuracy not only the maximum transaction throughput, but also important intermediate statistics such as commit probability. More in detail, TAS achieves a remarkable average relative error (defined as $|real - prediction|/real$) on the predicted throughput of 2%, with a maximum of 3.5%.

Comparison with a pure ML approach. We conclude by comparing the accuracy of TAS with that of a pure ML-based solution, namely the approach at the basis of several recent works in the area of elastic scaling [7, 15]. To this end, we trained Cubist on the TPCC-R workload, varying the number of nodes from 2 to 20 and the incoming load from 100 requests per second until reaching the maximum throughput. The input features for the ML included CPU, memory and network utilization, the percentage of update transactions and the mean number of locks they request, the transaction arrival rate, number of nodes and active threads per node. As in the previous evaluation study, we use maximum throughput as the output variable. These experiments were performed using Amazon EC2.

As test dataset, we use TPCC-W, which, we recall, generates a significantly higher data contention level with respect to TPCC-R. Further, unlike TPCC-R, TPCC-W exhibits a non-linear scalability trend. As expected [7], in these conditions, the pure ML-based approach manifests its limits in terms of reduced extrapolation power. In fact, the plots in Figure 6 clearly highlight that the pure ML-based solution tends to mimic the linear scalability trend that it observed during its training phase. As a consequence, it blunders when faced with workloads, like the TPCC-W, that i) have previously unobserved input characteristics, and ii) exhibit significantly different performance trends. This problem might be, to some extent, addressed by increasing the coverage of the training phase. However, achieving a good accuracy across a wide range of workloads may require a prohibitive increase of the ML training time. In fact, data contention dynamics in a (distributed) transactional system are influenced by a wide range of parameters [12, 3], and it is well known that the training time of ML techniques grows exponentially with the number of input features (the, so called, curse of dimensionality problem [5]).

The AM employed by TAS, on the other hand, can exploit the *a priori* knowledge on the dynamics of data consistency mechanisms to achieve a higher extrapolation power. Further, it allows to narrow the scope of (and hence to simplify) the problem tackled via ML techniques, reducing the dimensionality of the ML input features' space and, consequently, the duration of the training phase.

As a final remark, it is noteworthy to highlight that, in all our experiments, the performance attained with or without the monitoring framework were indistinguishable. Also, the time required to instantiate and solve a 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 automate elastic scaling.

6. CONCLUSIONS AND FUTURE WORK

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.

TAS relies on a novel hybrid forecasting methodology that jointly utilizes analytical modeling and machine learning techniques according to a divide-and-conquer approach: availability of precise knowledge of the concurrency control scheme/replication protocol is exploited to derive a white-box analytical model of data contention; black-block statistical techniques are instead used to capture the effects of contention on physical resources (CPU, memory, network) while avoiding explicit modeling 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 non-viable 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 deployed both on a private cluster and on a public cloud infrastructure (Amazon EC2).

Future work will be aimed at integrating analytical models for different concurrency control schemes in TAS and in extending TAS' lightweight data access pattern characterization to encompass also the case of partially replicated data sets.

Acknowledgment

This work has been partially supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/102496/2008 and PEst-OE/EEI/LA0021/2011, by the Cloud-TM project (co-financed by the European Commission through the contract no. 57784) and by COST Action IC1001 EuroTM.

7. REFERENCES

[1] M. Bennani and D. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proc. of the International Conference on Autonomic Computing (ICAC)*, 2005.

[2] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ansi sql isolation levels. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, 1995.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. 1986.

[4] U. N. Bhat, M. Shalaby, and M. J. Fischer. Approximation techniques in the solution of queueing problems. *Naval Research Logistics Quarterly*, 1979.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2007.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[7] J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content web servers. In *Proc. of the International Conference on Autonomic Computing (ICAC)*, 2006.

[8] B. Ciciani, D. M. Dias, and P. S. Yu. Analysis of replication in distributed database systems. *IEEE Transactions on Knowledge and Data Engineering*, 2(2), 1990.

[9] Y. Dai, Y. Luo, Z. Li, and Z. Wang. A new adaptive cusum control chart for detecting the multivariate process mean. *Quality and Reliability Engineering International*, 27(7), 2011.

[10] P. di Sanzo, B. Ciciani, F. Quaglia, and P. Romano. A performance model of multi-version concurrency control. In *Proc. of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2008.

[11] P. di Sanzo, B. Ciciani, F. Quaglia, and P. Romano. Analytical modelling of commit-time-locking algorithms for software transactional memories. In *Proc. of the International Computer Measurement Group Conference (CMG)*, 2010.

[12] P. di Sanzo, R. Palmieri, B. Ciciani, F. Quaglia, and P. Romano. Analytical modeling of lock-based concurrency control with arbitrary transaction data access patterns. In *Proc. of WOSP/SIPEW International Conference on Performance Engineering (ICPE)*, 2010.

[13] D. Dice, O. Shalev, and N. Shavit. Transactional locking ii. In *Proc. of the International Symposium on Distributed Computing (DISC)*, 2006.

[14] D. Didona, P. Romano, S. Peluso, and F. Quaglia. Transactional auto scaler: Elastic scaling of in-memory transactional data grids. Technical Report 50/2011, INESC-ID, December 2011.

[15] S. Ghanbari, G. Soundararajan, J. Chen, and C. Amza. Adaptive learning of metric correlations for temperature-aware database provisioning. In *Proc. of the International Conference on Autonomic Computing (ICAC)*, 2007.

[16] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, 1996.

[17] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proc. of the ACM Symposium on Cloud Computing (SOCC)*, 2011.

[18] L. Kleinrock. *Theory, Volume 1, Queueing Systems*. 1975.

[19] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Operating System Review*, 44, 2010.

[20] J. D. C. Little. A proof for the queuing formula: L= λ w. *Operations Research*, 9(3), 1961.

[21] D. A. Menascé and T. Nakanishi. Performance evaluation of a two-phase commit based protocol for ddbs. In *Proc. of the ACM SIGACT-SIGMOD symposium on Principles of Database Systems (PODS)*, 1982.

[22] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proc. of the ACM European Conference on Computer Systems (EuroSys)*, 2010.

[23] 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 the ACM European conference on Computer Systems (EuroSys)*, 2009.

[24] J. R. Quinlan. *C4.5: Programs for Machine Learning*. 1993.

[25] Red Hat / JBoss. JBoss Infinispan. `http://www.jboss.org/infinispan`.

[26] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proc. of the ACM Symposium on Cloud Computing (SOCC)*, 2011.

[27] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. of the International Conference on Autonomic Computing (ICAC)*, 2010.

[28] Y. C. Tay, N. Goodman, and R. Suri. Locking performance in centralized databases. *ACM Transactions on Database Systems*, 10, 1985.

[29] S. Thompson. *Sampling*. 2002.

[30] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. A. B. Fortes. Fuzzy modeling based resource management for virtualized database systems. In *Proc. of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2011.

[31] Z. Wang, X. Zhu, and S. Singhal. Utilization and slo-based control for dynamic sizing of resource partitions. In *Proc. of IFIP/IEEE Distributed Systems: Operations and Management (DSOM)*, 2005.

[32] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, 1995.

[33] 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 *Proc. of the ACM Symposium on Cloud Computing (SOCC)*, 2011.

[34] P. S. Yu, D. M. Dias, and S. S. Lavenberg. On the analytical modeling of database concurrency control. *Journal of the ACM (JACM)*, 40, 1993.

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