Self-Tuning Transactional Data Grids: the Cloud-TM Approach

Diego Didona
INESC-ID / Instituto Superior Técnico
didona@gsd.inesc-id.pt

Paolo Romano
INESC-ID / Instituto Superior Técnico
romano@inesc-id.pt

***Abstract*—In this paper we focus on the problem of self-tuning distributed transactional cloud data stores by presenting an overview of the autonomic mechanisms integrated in the Cloud-TM platform, a transactional cloud data store developed in the context of a recent European project.**

Cloud-TM takes a holistic approach to self-tuning and elastic scaling, treating them as strongly intertwined problems with the ultimate goals of i) achieving optimal efficiency at any scale of the platform, and ii) minimizing resource consumption in presence of varying workloads. From a methodological perspective, this is achieved by relying on the innovative idea of exploiting the diversity of different modelling approaches, including analytical models, machine-learning and simulations. By employing these modelling techniques in synergy, the Cloud-TM platform can dynamically optimize the underlying distributed data store over a number of dimensions, including its scale, the strategy it adopts to distribute and replicate data among the platforms' nodes, as well as its replication protocol.

I. Introduction

The appearance of the first commercial Cloud Platforms has represented a step towards the materialization of the Utility Computing vision. In typical Cloud Infrastructure as a Service (IaaS) platforms, in fact, resources are dispensed elastically, with a seemingly unbounded amount of computational power and storage available on demand, in a pay-only-for-what-you-use pricing model. This *elastic scaling* capability comes with the promise of enormous money saving and efficiency, but at the same time it poses two major challenges. On one hand, developers are faced with the non-trivial task of building distributed applications tailored for a very dynamic, elastic and fault-prone environment; on the other, in order to take advantage of the pay-for-what-you-use pricing model, system's administrators have the burden of determining, from time to time, the optimal configuration and scale for the platform, depending on the workload faced by the deployed applications.

Recently, we are witnessing the proliferation of a new breed of Cloud Data Stores, specifically built to tackle these issues. On one hand they provide abstractions and programming paradigms aimed at aiding the developers in facing the complexity of accessing and manipulating the application's distributed state. On the other, they come with primitives and mechanisms aimed at supporting elastic scaling of the platform and facilitating automatic resource provisioning.

Though built for the same target, a plethora of different solution have been proposed, both in industry and academia, exploring several trade-offs in the vast design space of distributed data management schemes. This heterogeneity, witnessed also in decades of research in the field of distributed data platforms, is symptomatic of the non-existence of a universal one-size-fits-all data management solution that maximizes the efficiency of a platform in all possible scenarios. The effectiveness of every scheme, in fact, strongly depends on two - dynamic - factors: i) the characteristics of the incoming workload, such as its intensity, the ratio of read/write operations, as well as the spatial/temporal locality in the data access patterns, and ii) the scale of the system.

Cloud-TM [1] is a recently concluded FP7 European project that developed a data-centric Platform as a Service (PaaS) aimed at maximizing ease of programming, while minimizing human intervention in the process of resource provisioning and runtime optimization of the platform. Most data platforms for virtualized environments, both in industry and academia[2], [3], [4], typically ensure weak consistency [5] models, favouring scalability over ease of application development. Cloud-TM, conversely, ensures strong consistency by leveraging on the abstraction of *transaction* [6]. Transactional consistency and scalability, two properties often seen as antagonists, are reconciled in the Cloud-TM platform thanks to innovative transactional consistency schemes [7], [8] designed precisely to meet the scalability and elasticity requirements of typical cloud infrastructures.

In this paper, we shall focus on the autonomic capabilities of the Cloud-TM platform, presenting the architecture of the module that is in charge of driving the self-tuning and resource provisioning of the Cloud-TM platform, i.e., the Cloud-TM Adaptation Manager. Most industrial and academic cloud data stores include supports for elastic scaling, and several also include mechanisms for automating the resource provisioning process. However, the self-tuning of any other platforms' parameters is at least uncommon for commercial platforms (at least based on available documentation [9], [10]). On the other hand, academic solutions for self-tuning of cloud data stores [11], [12], [13] target individual platform's parameters, failing to capture the strong intertwining between them (e.g., the choice of the replication protocol is clearly dependent on the platform's scale and on the number of data replicas stored in the platform). The Cloud-TM platform, conversely, takes a unique approach to self-tuning, based on two principles:

- **Multi-dimensional self-tuning.** The parameters space of a transactional data platform is quite vast. Moreover, the effect that these parameters have on performance are often intertwined, i.e., optimizing only with respect to a subset of them will lead to a suboptimal global configuration. As such, in order to globally optimize application's performance while minimizing operational costs, the Cloud-TM platform employs pervasive self-tuning schemes, which act at each layer of

This work has been partially supported by the projects "Cloud-TM" (co-financed by the European Commission through the contract no. 257784) and specSTM (PTDC/EIA-EIA/122785/2010) and by FCT (INESC-ID multiannual funding) through the PEst-OE/EEI/LA0021/2013 Program Funds.

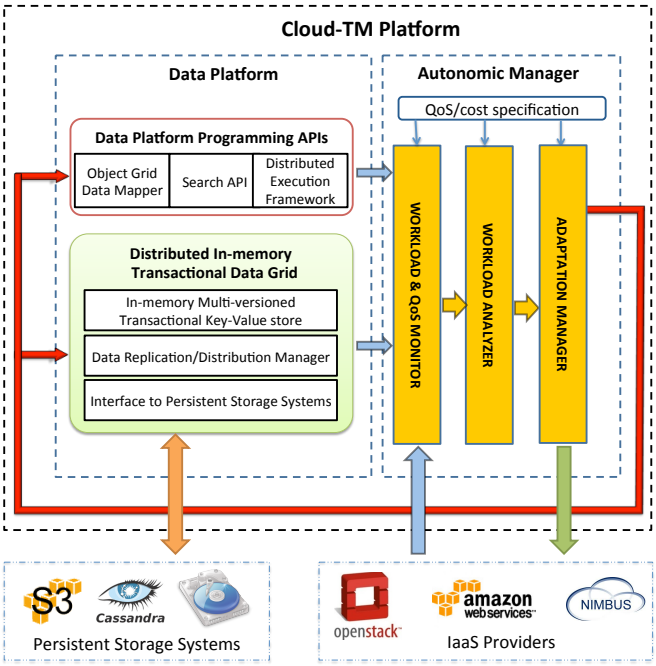


Fig. 1: Cloud-TM high-level architecture.

the architecture, and explicitly take into account the complex interdependencies among the different parameters.

• **Model diversity.** Given the multi-dimensional optimization space and the heterogeneity of transactional workloads, identifying the best configuration for a distributed data platform is a very complex task. Throughout the years, different techniques have been employed to predict the performance of a distributed application. Unfortunately, every performance prediction methodology has both strong and weak points, thus being unable, alone, to accurately predict applications' performance for each workload and for each platform configuration. For this reason, the Cloud-TM platform employs several, orthogonal modelling techniques, that are combined together in order to reliably identify the optimal configuration for the platform for any workload and respecting budget constraints.

The remainder of the paper is structured as follows: Sec. II provides a high-level overview of the Cloud-TM platform; Sec. III presents the Autonomic Manager; Sec. IV overviews related work; Sec V concludes the paper.

II. Overview of the Cloud-TM platform

The high-level architecture of the Cloud-TM platform is depicted in Fig. 1. It consists of two main components: the Data Platform and the Autonomic Manager, which are described in the following.

A. The Data Platform

The Data Platform is the module responsible for storing, retrieving and manipulating the data composing the state of the application and consists, on its turn, of two sub-modules, namely the Data Platform Programming APIs and the Distributed in-memory Transactional Data Grid.

Data Platform Programming APIs. This module exposes a set of APIs that provide abstractions aimed at keeping the complexity of developing applications for large-scale elastic cloud infrastructures as low as in conventional/non-distributed environments. These APIs are exposed by three distinct components, which we overview in the following.

Object Grid Data Mapper. This module allows programmers to transparently store and manipulate object-oriented domain models. As we shall see shortly, the back-bone of the Cloud-TM Data Platform is represented by a transactional key-value data store. This module, hence, acts as a bridge between the Cloud-TM programming environment and the underlying distributed data store, exposing to developers a much more expressive programming paradigm and data model than classic key-value stores.

Search API. The Search API allows applications to define ad-hoc queries to retrieve and manipulate portions of the state that they manage. The Search API is fully integrated with the object-oriented programming paradigm of Cloud-TM, supporting intrinsic aspects of the object-oriented model, such as polymorphism and inheritance. This result is achieved by integrating some of the leading open-source projects in the area of data management and indexing, namely Hibernate ORM and Apache Lucene, and extending them in order to efficiently distribute the index across the underlying data store.

Distributed Execution Framework (DEF). The DEF exposes a set of abstractions aimed at simplifying the development of parallel and distributed applications, allowing ordinary programmers to take full advantage of the processing power available by the set of distributed nodes of the Cloud-TM Platform without having to deal with low level issues such as load distribution, threads synchronization, and fault-tolerance.

Distributed in-memory Transactional Data Grid. This component is in charge of maintaining application's data and of guaranteeing the consistent evolution of the application's state. Below we describe its key components.

In-memory Multi-Versioned Transactional Key-Value Store. The backbone of the Cloud-TM Data Platform is represented by Infinispan [14], a key-value store with support for transactions. In order to maximize performance, Infinispan maintains data in-memory, and achieves fault-tolerance via data replications rather than via disk logging mechanisms. To achieve strong consistency without sacrificing scalability, Infinispan integrates GMU [7], a novel, fully decentralized, multi-versioning algorithm that achieves high scalability by means of genuine replication techniques (which ensure that only the nodes replicating data accessed by the transaction are involved in its processing), and by avoiding to ever abort or block read-only transactions.

Reconfigurable Replication Manager. Transactional workloads are very heterogeneous and affected by so many variables that no-one-size-fits-all solution exists that guarantees optimal performance across all possible applications' workloads. To address this issue, the Cloud-TM Platform integrates several data replication strategies, which exhibit different trade-offs and are, consequently, optimized for different workloads. The platform is, moreover, able to efficiently switch from one protocol to another, minimizing service interruption and aborts of transactions during the commutation.

More in detail, the Cloud-TM platform integrates three replication protocols: i) a Two-Phase Commit (2PC) protocol, in which each node of the platform can execute both update

and read-only transactions, relying on locks acquisition at commit time to determine their outcome; ii) a Primary Backup (PB) protocol, in which only a node, namely the primary, is allowed to serve update transactions; iii) a Total-Order (TO) based protocol, which relies on a total order primitive to determine the outcome of a transaction. The pros and cons that comes with any of these protocols are discussed in [15].

Interface to Persistent Storage. The Cloud-TM Data Platform supports the possibility to persist its state over a wide range of heterogeneous durable storage systems, ranging from local/distributed file systems to cloud storages (e.g., Amazon's S3 [16] or Cassandra [3]). This capability serves a twofold purpose: i) it allows to cope with scenarios in which the amount of main memory affordable according to budget constraints is not enough to accommodate the whole application's state and ii) it guarantees the durability of the application's state even in spite of a failure/redeployment of the whole application, still giving the possibility to the user to shift the costly interaction with the stable storage out of the latency-sensitive path of transactions.

B. The Autonomic Manager

The second main building block of the Cloud-TM platform is the Autonomic Manager, which is the component in charge of the self-tuning of the Data Platform. Its architecture is concisely depicted in Fig. 1 and further detailed in Fig 2. The Autonomic Manager is formed by three main subsystems, called QoS Specification Module, Workload and Performance Monitor (WPM), Workload Analyzer (WA) and Adaptation Manager (AdM). Further, a QoS Specification Module

In the following we are going to overview briefly the functionalities of the WPM, WA and QoS specification modules; the AdM will be described in higher detail in the next section.

Quality-of-Service specification module. This module is responsible for recording the desired QoS levels for the application and for contrasting them at runtime with the measured performance of the platform, in order to ensure that they are matched and trigger notifications if they are not. Currently, this module allows the QoS levels to be formalized according to the models developed in the context of the European project SLA@SOI [17].

Workload and Performance Monitor. The WPM is the subsystem in charge of gathering statistical information on the workload and performance/efficiency of the various components/layers of the Cloud-TM Platform, and of conveying them towards the WA and the AM. Examples of the performance metrics collected by the WPM are the application throughput, transactions response time and abort rate.

The set of collected application-level statistics is quite broad, and allows to perform a very accurate runtime workload characterization: other than collecting average values for statistics like read-only vs update transactions ratio and number of operations per transaction, the WPM also keeps tracks, by means of lightweight probabilistic techniques [18], of data items that represent hot-spots for what concerns data contention and/or data locality. This information is then exposed by the WA to the user, so that she can get a feedback on the hot-spots in the application's data access pattern, and, as we shall see later, it is also fed to the AdM, which exploits it to detect and correct sub-optimal mappings of data onto nodes.

As we will see, the WA exploits the monitoring data streams produced by the WPM also to automatically detect shifts of the workload that may give raise to QoS violations and/or lead the Cloud-TM Data Platform to operate in sub-optimal configurations. This information is exploited, in its turn, by the AdM, which can react triggering corrective actions aimed at altering the scale and/or configuration of the Data Platform. More details about the architecture of the WPM and the workload characterization can be found in [19], [20].

Workload Analyzer. The WA acts as an intermediary between the WPM and AdM, and bears three different responsibilities.

First, it is responsible for data aggregation: the streams of monitoring data produced by the distributed nodes of the Cloud-TM Platform via the WPM are gathered by the WA, which exposes programmatic APIs and web-based GUIs allowing for aggregating statistics originated by different software layers and/or groups of nodes.

In the second place, the WA is in charge of performing data filtering and workload/KPI change detection: the WA integrates algorithms aimed at detecting statistically relevant variations of platform's KPIs and/or workload characteristics. These techniques allow filtering unavoidable statistical fluctuations and enhance the stability and robustness of the self-tuning mechanisms integrated in the AdM.

Finally, the WA integrates workload and resource demand prediction schemes: the WA includes algorithms for time-series forecasting (e.g., based on Kalman filter or on polynomial regression [19]), which allow predicting future workload's trends and allow the AdM to enact proactive self-tuning schemes. This functionality represents a fundamental building block for any proactive adaptation scheme, i.e., schemes triggering reconfigurations of the platform anticipating imminent workloads' changes, which are particularly desirable in case the platform's reconfiguration (as in the case of elastic scaling) can have non-negligible latencies.

In the next section we are going to present in detail the core module of the AM, namely the Adaptation Manager.

III. Adaptation Manager

The Adaptation Manager (AdM) is the key component of the AM. As already mentioned, this module is in charge of driving the self-tuning of a number of mechanisms of the Cloud-TM Data Platform, as well as of automating its QoS-based resource provisioning process.

Fig. 2 depicts the internal architecture of the AdM, highlighting its main building blocks and how it interacts with the other modules of the Cloud-TM Platform. The AdM is formed by two main subcomponents, namely the Performance Prediction Service (PPS) and the Platform Optimizer (PO).

The PPS encapsulates diverse performance forecasting mechanisms that rely on alternative predictive methodologies working in synergy to maximize the accuracy of the prediction system, and, consequently, of the whole self- tuning process. In more detail, the PPS exploits the notion of *model diversity*, i.e., it combines white-box (e.g., analytical models) and black-box (e.g., machine-learning techniques) approaches with complementary strengths and weaknesses in order to take the best of the two approaches, namely:

- the high accuracy of black-box statistical methods when faced with workloads similar to those witnessed during their training phase;
- the minimal training phase of white-box methods, and their high extrapolation power, i.e., their ability to achieve good

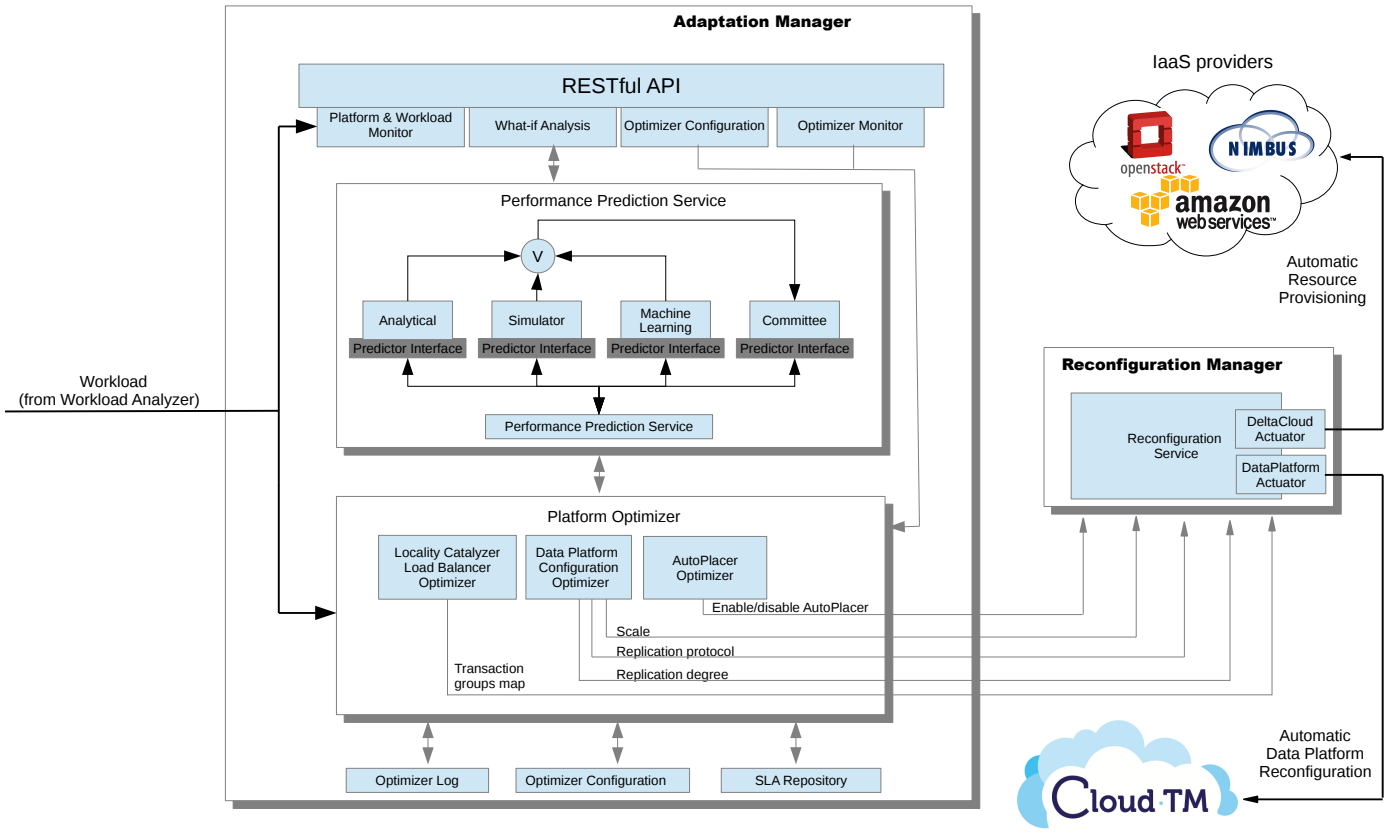


Fig. 2: Autonomic Manager architecture.

accuracy even when providing forecasts concerning previously unexplored regions of the workloads' parameter space.

The PPS is used not only to guide the optimization process, as we will discuss shortly, but also to allow the end-users to conduct what-if analysis aimed at assessing the performance achievable by the Cloud-TM Platform when deployed over platforms of different scales, and in presence of different workload types. This type of analysis can be extremely valuable for the developers of Cloud-TM applications, who can gain insights on the scalability and efficiency of their applications, speculating on the impact on performance due to alternative implementation designs and/or workload's shifts.

The PO, on the other hand, is the component in charge of defining the reconfiguration strategy of the various self-tuning schemes embedded in the Cloud-TM Platform. This module has a flexible an extensible software architecture, which allows specifying a chain of optimizers aimed at tuning different parameters/behaviours of the Data Platform, namely:

i) its scale, i.e, the number and type of nodes over which the Cloud-TM Data Platform is deployed;

ii) the number of data replicas, or, shortly, replication degree;

iii) the employed replication protocol (among the three currently supported, see Section II);

iv) the placement of data across the nodes of the platform.

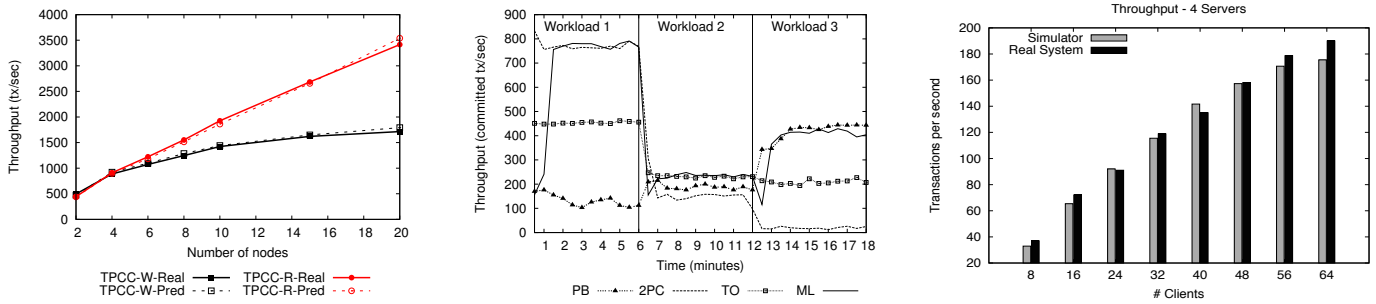
A noteworthy aspect of the PO is its high flexibility in supporting different degrees of automation of the system. To this end, the AM exposes REST-based interfaces that allow the monitoring and fine-grained configuration of its automatic adaptation policies. These REST APIs are exploited by the Monitoring and Administration Web Console (not described in the paper for space constraints), which allows human end-users to specify which of the available self-tuning mechanisms should be fully automated by the Adaptation Manager, and which reconfigurations should only be recommended and performed after the explicit approval of the platform's administrators. The Web Console serves also as an access point to the what-if analysis facilities, by providing a user friendly web interface to the forecasting capabilities of the PPS.

Finally, the complexity associated with enacting the reconfigurations decreed by the PO is encapsulated by the Reconfiguration Manager (RM). This module receives as input the set of adaptations specified by the PO and i) defines the order in which the optimizations will be performed, taking into account any possible dependencies that could affect the efficiency/correctness of the whole reconfiguration process; ii) coordinates a set of actuators that allow to enact the supported adaptations, hiding the complexity of interacting with heterogeneous types of resources/layers of the platform.

In the following we provide a more detailed description of the PPS, PO and RM.

A. *Performance Predictor Service*

As already mentioned, the PPS can leverage on diverse prediction methodologies, i.e., analytical models, machine learning techniques and simulation techniques. Before discussing how these techniques are jointly exploited in the AdM, we first provide additional details on their internals and capabilities.



(a) Analytical model's accuracy [21]. (b) ML-driven protocol switching [15]. (c) Simulative model's accuracy [22].

Fig. 3: Examples of utilization of the performance predictors

• **Analytical predictor.** This predictor [21] relies on a queueing theory-based mean-value analysis techniques to forecast the probability of transaction commit, the mean transaction response time, and the maximum system throughput. One key innovative element of the modeling approach is that it does not rely on classic assumptions on the uniformity of transaction's accesses over the whole data-set (which can be strongly limiting in presence of skewed access patterns). Instead, it introduces a powerful abstraction that allows the on-line characterization of the application data access pattern in a lightweight and pragmatical manner, i.e., a scalar value called Application Contention Factor (ACF). The key idea underlying the ACF is to capture the effects on data contention generated by arbitrary access pattern by means of a simpler *uniform* access pattern (over a data set having a possibly different size) generating an equivalent amount of data contention [21].

• **Simulative predictor.** This predictor is based on Discrete Event Simulation techniques, and includes a set of discrete event models for simulating the behavior of the different operating modes supported by the Cloud-TM Data Platform. The whole architecture of the simulation component is highly modular, since it is based on skeleton models [22], which allow the instantiation of actual models able to capture the dynamics of various distributed data management schemes and platform scales. Also, it is highly configurable, since it offers a suite of different embedded data access models, relying on a wide set of parametrizable distributions, and also offers the possibility to simulate data access patterns based on traces of the accesses, as provided by the tool chain formed by WPM and WA. These include information about the most accessed data, which can have a strong impact on performance especially in applications characterized by strongly skewed access patters, as it is, for instance, the case of well-known transactional benchmarks [23].

• **Machine Learning based predictor.** This predictor relies on pure black-box Machine Learning (ML) techniques to forecast the throughput, abort rate and mean execution time (or its x-th percentile) of the transaction classes composing the input workload. This is a multiple-input-multiple-output (MIMO) regression problem, in which for each of the above parameters, we aim at identifying a corresponding function that captures their dynamics over an input space composed by a rich set of features characterizing the workload and the scale/configuration of the platform. Such features are determined through the execution of an automated feature selection algorithm, which is aimed at minimizing the risk of overfitting and maximizing the generality of the model by discarding features that are either too closely correlated among each other (and hence redundant), or too loosely correlated with the output variable (and hence useless). Specifically, the machine-learner based predictor relies on the Forward Selection [24], [25] technique, a greedy heuristic that progressively extends the set of selected features till the accuracy it achieves when using 10-fold cross-validation on the training set is maximized.

Fig. 3 shows three possible applications of these tools, to predict the performance of a popular benchmark for transactional platform, namely TPC-C [23][1]. Fig. 3a shows the capabilities of the analytical model of forecasting the throughput achievable by the application when deployed over a platform of different sizes depending on its workload (TPCC-W stands for write intensive, TPCC-R for read intensive). Fig. 3b shows the capabilities of the machine learner of determining, for a fixed scale configuration, the best replication protocol in face of changing workloads, leading to the automatic optimization of this parameter. Finally, Fig. 3c shows the accuracy of the simulative predictor in forecasting the performance achievable when deploying TPC-C on a set of 4 VMs on Amazon EC2.

Given that these methodologies are highly complementary, their joint usage within Cloud-TM allowed the construction of an AM component supporting a wide spectrum of prediction (and hence optimization) capabilities.

As for the employment of machine learning, these techniques provide highly reliable predictions in system configurations (e.g., in terms of number of nodes within the underlying virtualized infrastructure) and workload profile/intensity falling within an already explored domain of the parameters' space. The exploration (and hence the associated training phase for the machine learner) can performed either off-line or on-line. Further, the training outcome can be refined along time. On the other hand, machine learning techniques are known to provide limited predictive power when working in extrapolation, i.e., when considering previously unseen system configurations. These issues are avoided, conversely, by analytical [21] and simulative models [22], thanks to the employment of white-box modelling techniques.

Concerning computational costs, both analytical and ma-

[1]Note that the original TPC-C benchmark is designed to operate on a relational database, hence we developed a porting running directly on top of a transactional key-value store such as Infinispan (code available here: http://github.com/cloudtm)

chine learning models (during the querying phase) are typically less demanding than simulative approaches. On the downside, the design phase of analytical models is normally significantly more onerous. Also, due to their inherently higher complexity, analytical approaches tend to introduce a number of simplifying assumptions, which may ultimately degrade their prediction accuracy. Regarding the simulation performance, the simulation framework has been developed using the ROOT-Sim high performance parallel simulation engine [26]. Hence, AM also entails high performance simulation capabilities, which make very large models tractable and solvable within significantly reduced computation time.

Finally, the PPS includes also predictors that combine the above techniques/methods in a synergic (rather than orthogonal) manner. Particularly, the PPS offers hybrid predictors that rely on analytical or simulative models to predict the effects of data contention, and on machine-learning techniques for estimating the network-bound latencies due to inter-node synchronization. To this end, the analytical/simulative models have been interfaced with Cubist [2], a decision tree regressor that approximates non-linear multivariate functions by means of piece-wise linear approximations. Cubist is used to provide estimations on the latencies of various types of network-bound operations, e.g., the latency to fetch data items remotely and of the commit schemes employed by the different replication protocols supported by the Cloud-TM platform.

Another approach for combining these alternative modelling techniques is to exploit voting schemes to combine the predictions output by the various forecasters, and weight them according to different strategies. The current AdM prototype supports a relatively simple voting scheme that averages the outputs of the single forecasters. We are currently pursuing a research line focused on devising more sophisticated schemes based on gating and boosting [27].

As hinted in Sec. II-B, the PPS is triggered whenever the WA detects statistically relevant variations of the set of metrics collected by the WPM. This allows to re-evaluate the optimality of the platform configuration in presence of shifts of the workload's characteristics, or upon variations of the data access pattern locality caused by the optimization of the data placement, or even upon variations of the costs associated with transactions processing due to a change of behavior of the underlying virtualized infrastructure. In a complementary fashion, it is possible to trigger the PPS manually via REST APIs, or to configure the PPS to be queried periodically.

B. Platform Optimizer

As already mentioned, the Optimizer is composed by two sub-optimizers, which are in charge of different self-tuning processes:

- The Data Platform Configuration Optimizer, which is in charge of automating the tuning of the platform's scale, degree of replication, and of the choice of its replication protocol.
- the AutoPlacer Optimizer, which monitors the quality of the current data placement policy, and orchestrates the execution of an algorithm aimed at maximizing data locality.

We overview these 2 optimizers in the following.

Data Platform Configuration Optimizer. The platform's scale (noted as s), the choice of the employed replication protocol (noted as r) and the data replication degree (noted as d) are three tightly intertwined parameters. For instance, full replication is often preferable in small scale systems (e.g., up to 10 machines). Conversely, partial replication is typically advisable for large scale systems, unless the workload is strongly read-dominated. Also, a primary backup replication protocol is likely to suffer from scalability issues in large scale system, due to the constraint of allowing a single node to process update transactions. On the opposite, in small scale systems, and when faced with conflict intensive workloads, the primary backup protocol is typically more efficient than multi-master replication protocols (such as 2PC or TO) which can incur in higher abort rates and generate a larger network traffic and hence longer commit latencies.

In order to tackle these issues, the self-tuning process of these three key parameters of the Cloud-TM Platform relies extensively on the set of performance predictors described in Sec. III. More precisely, the optimizer that is in charge of tuning these parameters determines their optimal values by means of a search in the three-dimensional parameters space ($s \times r \times d$). Each step of the search algorithm consists of a query to one of the predictors made available by the performance prediction service, which evaluates the quality of a given configuration of this triple of parameters. The quality of the considered configuration is evaluated on the basis of the predicted values for the KPIs defined in the QoS specification (typically throughput, response time and abort rate) and the operational cost of the considered configuration.

The search algorithm aims at identifying the configuration of this triplet of parameters that has minimum cost and that ensures that the desired QoS is met. The search algorithm can be configured to explore alternative configurations for any subset of these three parameters. This enhances the flexibility of the self-tuning mechanisms of the Cloud-TM Platform, allowing system administrators to impose constraints on which configuration's parameters should be statically assigned (i.e., not automatically adjusted) and which ones should rather be self-tuned by the AM.

The current prototype integrates a relatively simple search algorithm that performs an exhaustive search in the parameters' space to determine the optimal configuration given the current workload and desired QoS. In addition to its simplicity, the key advantage of this approach is that it ensures that the optimal configuration (according to the forecasts of the PPS) can always be identified. On the other hand, for very large scale systems, the growth of the parameters' space may hinder the efficiency of this optimizer. To cope with this issue, we designed the software architecture of this optimizer to be readily extended to incorporate classic search heuristics (such as hill climbing or simulated annealing [27]) that trade off completeness (and hence sacrifice optimality guarantees) to maximize efficiency.

AutoPlacer Optimizer. In a distributed data platform, such as Cloud-TM, processing applications' requests can imply accessing data that is stored remotely, i.e., on different nodes of the platform. Hence, the performance and scalability achievable by applications can be affected by the quality of the algorithms used to distribute data among the nodes of the platforms. These should be accurate enough to guarantee high data locality (and minimize remote data accesses), as well as sufficiently lightweight and scalable to cope with large scale applications. AutoPlacer addresses this problem by automatically identifying the data items having a sub-optimal placement onto

[2] `http://www.rulequest.com`

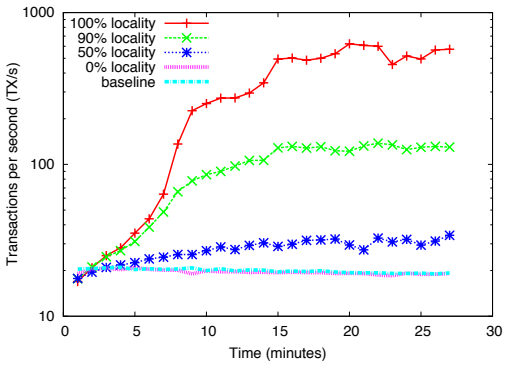


Fig. 4: Impact of AutoPlacer on application's throughput.

the platform and re-locating them automatically to maximize access locality. Scalability and practical viability of AutoPlacer are achieved via innovative probabilistic algorithms, which exploit stream-analysis and machine-learning techniques [18]. Fig. 4 depicts the impact of the AutoPlacer algorithm on the throughput of our porting of TPC-C benchmark initially deployed with a random data placement. It is possible to see that, if the application exhibits no locality, then the overhead imposed by AutoPlacer is negligible; conversely, if the application does exhibit locality in the data access pattern, its throughput can be improved, round after round, up to a factor of 50x, depending on the degree of locality.

C. Reconfiguration Manager

The RM is in charge of actuating the set of reconfigurations decided by the AM. More in detail, this module receives as input the set of adaptations specified by the PO and performs the two following steps: i) it establishes the order in which the various specified reconfigurations will be enacted, and ii) accordingly coordinates the execution of such reconfigurations by means a set of actuators that allow to acquire/release resource from different IaaS providers as well as with to interact with the various layers of the Cloud-TM Platform that support dynamic reconfiguration.

Concerning the first point, it has to be noted that the order in which certain platform's reconfigurations are executed can impact the reconfiguration latency and its effectiveness. For instance, it is highly desirable that reconfigurations involving alterations of both the platform's scale and its replication degree are executed concomitantly. If this two reconfigurations were executed separately, in fact, the data hosted by the platform is likely to be transferred more than once among the nodes of the platform (a first time to distribute data over the new set of nodes, and a second one upon change of the replication degree).

The RM encapsulates the logic in charge of orchestrating the set of adaptations requested by the AM, relying on a rule based system that encodes the set of dependencies among the set of possible reconfigurations supported by the AM. The actual reconfiguration of the various components/modules of the Cloud-TM Platform is managed by two type of actuators.

i) *δ-cloud actuator.* This actuator is in charge of managing the interactions with a number of IaaS cloud providers. To maximize portability, we rely on the Apache δ-cloud project [28]. δ-cloud provides a Cloud abstraction API that works as wrapper around a large number of Clouds, abstracting their differences.

ii) *Cloud-TM Data Platform Actuator.* This actuator is in charge of triggering reconfigurations that involve layers of the Cloud-TM Data Platform, such as, activating the AutoPlacer scheme, changing the data replication degree or the replication protocol, etc. In order to uniformize interfaces, and maximize portability, it relies industry standard JMX (Java Management Extensions) technology to expose the methods that trigger the various types of supported reconfiguration.

IV. Related Work

The exploitation of model diversity, more specifically the combination of machine learning and analytical models, has already been proposed in a number of fields. In [29], a model based on queuing theory is corrected at runtime by exploiting online reinforcement learning to determine the batching level delivering lowest latencies for a Total-Order based broadcast primitive. A similar approach is undertaken in [30], where the target is optimizing resource provisioning in a distributed application and the online learner is based on Q-learning. In [31], [32] analytical models are complemented at runtime by decision tree regressors, in the former case with the purpose of optimizing the global multiprogramming level for distributed transactional applications, in the latter to allow a continuous validation and correction of difference performance predictors in a data center. Though similar in the spirit, those works focus on a single aspect of a specific platform (e.g., multiprogramming level) or on several orthogonal aspects of a whole data center [32]. To the best of our knowledge, the Cloud-TM platform is the first example of application of exploitation of model diversity to solve a multi-dimensional optimization problem, in which it is not possible to find the global optimum by separately performing optimization on single dimensions.

Also the set commercial solutions and the body of literature regarding autonomic schemes for resource provisioning and tuning of distributed data platform are very wide. As already stated in Sec.I, commercial and open-source platforms tailored for virtualized environments focus on providing supports for automatic resource provisioning. Platforms like Google App Engine [9], Microsoft Azure [10] and frameworks like Amazon AutoScaling [33] and OpenNebula [34] provide a set of APIs that have to be implement in order to specify the provisioning policy. Such policy, however, can be typically defined only in terms of simple rules based on the measurement of simple system metrics (e.g., CPU utilization); the user is, thus, left with the burden of the non-trivial task of devising the elastic scaling rules as well as the time-consuming task of collecting application-specific statistics.

Academic proposals exploit a very wide range of techniques to automatize the resource provisioning process without human intervention. However, they all come with the limitations that are proper of the employed techniques: [35], [36] exploit queuing theory, but avoiding to explicitly modeling data contention; [37], [38], [39] rely only on machine learning, thus being prone to poor effectiveness when facing previously unseen workloads.

The two solutions that are more related to the Cloud-TM pervasive self-tuning approach are MeT [11] and the SCADS Director [4]. The former performs resource provisioning while also clustering data accessed by a similar workload onto nodes in the platform whose configuration is optimized for that workload. However, this solution requires a preliminary definition of the considered workloads and of the best configuration

for nodes serving them; moreover, the provisioning scheme is model-free, iteratively acquiring and releasing resources until the system's scale is optimized for the incoming workload. The SCADS director, conversely, exploits a feedforward controller, whose inner model is built via regression, in order to provision a key-value store, move data and change the replication degrees of subsets of data. The goal is keeping the 99-th percentile of response on single operations under a predefined threshold. Being based on pure machine learning, the SCADS director cannot perform what-if analysis, other than being liable for the aforementioned limitations of pure machine learning. Moreover, it is worth mentioning that this solution is tailored for an eventual consistent platform, in which the inter-dependencies among replication degree, data placement and scale are weaker than in transactional platforms like Cloud-TM.

Regarding the replication protocol, we are aware of only two solutions that encompass its self-tuning, namely HTR [12] and PolyCert [13]. In both, the most appropriate protocol to validate the transaction with is determined on a per-transaction basis. However, the former requires the user to implement the oracle responsible for selecting the best choice; the latter relies on a combination of off-line and online statistical learning, thus being prone to the already discussed limitations.

V. Conclusions

In this paper we presented an overview of the self-tuning techniques integrated in Cloud-TM, a transactional cloud data store developed in the context of a recent European project. Cloud-TM takes a holistic approach to self-tuning and elastic scaling, treating them as strongly intertwined problems with the ultimate goals of i) achieving optimal efficiency at any scale of the platform, and ii) minimizing resource consumption in presence of varying workloads. From a methodological perspective, this is achieved by relying on the innovative idea of exploiting the diversity of different modelling approaches, including analytical models, machine-learning and simulations.

By employing these modelling techniques in synergy, the Cloud-TM platform can dynamically optimize the underlying distributed data store over a number of dimensions, including its scale, the strategy it adopts to distribute and replicate data among the platforms' nodes, as well as its replication protocol.

References

[1] P. Romano, L. Rodrigues, N. Carvalho, and J. Cachopo, "Cloud-TM: Harnessing the Cloud with Distributed Transactional Memories," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, Apr. 2010.

[2] Amazon, "Dynamodb," http://aws.amazon.com/dynamodb/.

[3] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, Apr. 2010.

[4] B. Trushkowsky, P. Bodík, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "The scads director: Scaling a distributed storage system under stringent performance requirements," in *FAST*, 2011.

[5] A. Adya, "Weak consistency: A generalized theory and optimistic implementations for distributed transactions," Tech. Rep., 1999.

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

[7] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, and L. Rodrigues, "When scalability meets consistency: Genuine multiversion update-serializable partial data replication," in *ICDCS*, 2012.

[8] S. Peluso, P. Romano, and F. Quaglia, "Score: A scalable one-copy serializable partial replication protocol," in *Middleware*, 2012.

[9] Google, "Google appengine," https://appengine.google.com/.

[10] Microsoft, "Azure," http://www.windowsazure.com/.

[11] F. Cruz, F. Maia, M. Matos, R. Oliveira, J. Paulo, J. Pereira, and R. Vilaça, "Met: workload aware elasticity for nosql," in *EuroSys*, 2013.

[12] T. Kobus, M. Kokocinski, and P. T. Wojciechowski, "Hybrid replication: State-machine-based and deferred-update replication schemes combined," in *ICDCS*, 2013.

[13] M. Couceiro, P. Romano, and L. Rodrigues, "Polycert: Polymorphic self-optimizing replication for in-memory transactional grids," in *Middleware*, 2012.

[14] F. Marchioni and M. Surtani, *Infinispan Data Grid Platform*. Packt Publishing, 2012.

[15] M. Couceiro, P. Ruivo, P. Romano, and L. Rodrigues, "Chasing the optimum in replicated in-memory transactional platforms via protocol adaptation," in *DSN*, 2013.

[16] Amazon, "S3," http://aws.amazon.com/s3/.

[17] SLA@SOI Consortium, "Sla@soi," http://sla-at-soi.eu.

[18] J. Paiva, P. Ruivo, P. Romano, and L. Rodrigues, "Autoplacer: scalable self-tuning data placement in distributed key-value stores," in *Proc. ICAC 2013*, 2013.

[19] B. Ciciani, D. Didona, P. Di Sanzo, R. Palmieri, S. Peluso, F. Quaglia, and P. Romano, "Automated workload characterization in cloud-based transactional data grids," in *IPDPSW*, 2012.

[20] R. Palmieri, P. di Sanzo, F. Quaglia, P. Romano, S. Peluso, and D. Didona, "Integrated monitoring of infrastructures and applications in cloud environments," in *Euro-Par*, 2011.

[21] D. Didona, P. Romano, S. Peluso, and F. Quaglia, "Transactional auto scaler: Elastic scaling of in-memory transactional data grids," in *ICAC*, 2012.

[22] P. Di Sanzo, F. Antonacci, B. Ciciani, R. Palmieri, A. Pellegrini, S. Peluso, F. Quaglia, D. Rughetti, and R. Vitali, "A framework for high performance simulation of transactional data grid platforms," in *SIMUTools*, 2013.

[23] The TPC Council, "Tpc-c," www.tpc.org/tpcc/.

[24] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, Mar. 2003.

[25] M. Couceiro, P. Romano, and L. Rodrigues, "A machine learning approach to performance prediction of total order broadcast protocols," in *SASO*, 2010, pp. 184–193.

[26] A. Pellegrini, R. Vitali, and F. Quaglia, "The rome optimistic simulator: Core internals and programming model," in *SIMUTools*, 2011.

[27] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[28] The Apache Foundation, "Deltacloud," deltacloud.apache.org/.

[29] Paolo Romano, Matteo Leonetti, "Self-tuning Batching in Total Order Broadcast Protocols via Analytical Modelling and Reinforcement Learning," in *ICNC*, 2012.

[30] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "On the use of hybrid reinforcement learning for autonomic resource allocation," *Cluster Computing*, vol. 10, no. 3, pp. 287–299, Sep. 2007.

[31] D. Didona, P. Felber, D. Harmanci, P. Romano, and J. Schenker, "Identifying the optimal level of parallelism in transactional memory applications," *Computing*, 2014.

[32] E. Thereska and G. R. Ganger, "Ironmodel: Robust performance models in the wild," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, Jun. 2008.

[33] Amazon, "Cloudwatch," http://aws.amazon.com/cloudwatch/.

[34] OpenNebula, "Opennebula," http://opennebula.org.

[35] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads," in *ICAC*, 2010.

[36] U. Sharma, P. Shenoy, and D. F. Towsley, "Provisioning multi-tier cloud applications using statistical bounds on sojourn time," in *ICAC*, 2012.

[37] J. Chen, G. Soundararajan, and C. Amza, "Autonomic provisioning of backend databases in dynamic content web servers," in *ICAC*, 2006.

[38] S. Ghanbari, G. Soundararajan, J. Chen, and C. Amza, "Adaptive learning of metric correlations for temperature-aware database provisioning," in *ICAC*, 2007.

[39] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris, "Automated, elastic resource provisioning for nosql clusters using tiramola," in *CCGRID*, 2013.