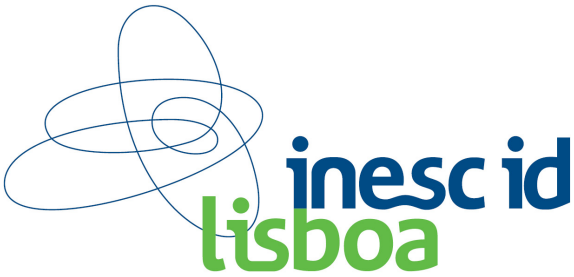

Technical Report RT/13/2014

Cloud-TM: Transactional, Object-oriented, Self-tuning Cloud Data Store

Paolo Romano
INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal

Mark Little
Red Hat, United Kingdom

Francesco Quaglia
CINI/Sapienza, Italy

Luis Rodrigues
INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal

Vittorio Amos Ziparo
Algorithmica, Italy

May 2014

Abstract

Cloud-TM is a highly innovative data-centric middleware platform, aimed at easing development and abating operational and administration costs of cloud applications.

The Cloud-TM platform is designed from the grounds up to meet the scalability and dynamicity requirements of cloud infrastructures, and is the result of a 3 years long collaboration between 2 academic and 2 industrial partners.

Cloud-TM provides intuitive, yet powerful abstractions aimed at masking complexity and at allowing ordinary programmers to unleash the potentiality of large-scale cloud platforms. Further, Cloud-TM integrates pervasive self-tuning schemes that exploit in a synergic way diverse methodologies like analytical modeling, simulation, and machine learning, to pursue optimal efficiency at any scale, and for any workload.

Cloud-TM: Transactional, Object-oriented, Self-tuning Cloud Data Store

Paolo Romano, IST/INESC-ID, Portugal
Mark Little, Red Hat, UK
Francesco Quaglia, CINI/Sapienza, Italy
Luis Rodrigues, IST/INESC-ID, Portugal
Vittorio Amos Ziparo, Algorithmica, Italy

Abstract

Cloud-TM is a highly innovative data-centric middleware platform, aimed at easing development and abating operational and administration costs of cloud applications.
The Cloud-TM platform is designed from the grounds up to meet the scalability and dynamicity requirements of cloud infrastructures, and is the result of a 3 years long collaboration between 2 academic and 2 industrial partners.
Cloud-TM provides intuitive, yet powerful abstractions aimed at masking complexity and at allowing ordinary programmers to unleash the potentiality of large-scale cloud platforms. Further, Cloud-TM integrates pervasive self-tuning schemes that exploit in a synergic way diverse methodologies like analytical modeling, simulation, and machine learning, to pursue optimal efficiency at any scale, and for any workload.

1. The Challenge

The appearance of the first commercial cloud computing platforms has represented a significant step towards the materialization of the vision of utility-computing. However, the promise of infinite scalability catalyzing much of the recent interest about cloud computing is still menaced by one major pitfall: the lack of programming paradigms and abstractions capable of bringing the power of parallel/distributed programming into the hands of ordinary programmers.

One of the most crucial issues to tackle when developing large-scale distributed applications is related to ***how to ensure consistency of the application's state in presence of concurrent manipulations.*** The challenge here is to identify mechanisms capable of ensuring adequate consistency levels while being:

1. simple and familiar for the programmers;
2. highly efficient and scalable;
3. fault-tolerant and highly available.

Decades of literature and field experience in areas such as replicated databases, Web infrastructures, and high performance computing have brought to the development of a plethora of different approaches to ensure state consistency in distributed platforms.

Unfortunately, the design space of distributed state consistency mechanisms is so vast that ***no universal, one-size-fits-all solution exists***, as the efficiency of individual state management approaches is strongly affected by:

1. the characteristics of the incoming workload, such as the ratio of read/write operations, as well as the spatial/temporal locality in the data access patterns, and
2. the scale of the system (e.g., small vs large number of nodes, local vs geographical distribution) on which these mechanisms are deployed.

The complexity of this problem is particularly exacerbated in Cloud Computing platforms due to a key feature of the cloud: its ability to elastically acquire or release resources, de facto dynamically varying the scale of the platform to meet the demands of varying workloads.

2. The Cloud-TM Approach

The Cloud-TM project addressed precisely the aforementioned issues by building a **highly innovative data-centric middleware platform**. The Cloud-TM platform is designed from the grounds up to meet scalability and dynamicity requirements, while providing intuitive, yet powerful abstractions aimed at masking complexity and allowing ordinary programmers to unleash the potentiality of large-scale cloud platforms.

Most cloud oriented solutions embrace weak consistency models that achieve scalability at the price of an increase of complexity for the programmers. This leads to a significant growth of software development costs and of the time to market, ultimately hindering competitiveness.

Conversely, Cloud-TM adopts an intuitive, yet scalable programming paradigm, enriched by a number of abstractions aimed at masking complexity. The Cloud-TM programming paradigm integrates the intuitive abstraction of **atomic transaction** as a first-class programming construct, sheltering programmers from having to deal with the idiosyncrasies of weak consistency models. Strong-consistency and scalability, two properties often seen as antithetic, are reconciled thanks to innovative transactional consistency schemes designed precisely to meet the scalability and elasticity requirements of typical cloud infrastructures.

Beyond transactional consistency, the Cloud-TM programming model provides transparent support for object orientation and queries, concurrency-friendly data structures, and frameworks to control the distributed execution of tasks, hiding issues such as fault-tolerance, load distribution and data placement.

Finally, Cloud-TM pursues the minimization of the other major source of costs for cloud-based applications, namely operational costs. This is achieved by means of two main mechanisms:

- **Automation** of the provisioning of cloud resources, based on user specified target criteria in terms of both Quality of Service and budget constraints. This allows guaranteeing that applications only use the

minimum amount of resources necessary to withstand the current load pressure.

- **Maximization of efficiency** (i.e., the benefits/costs ratio according to the cloud computing usage-based pricing model) by means of pervasive self-tuning schemes that adapt the middleware's internals to ensure optimal performance at any scale, and for any workload.

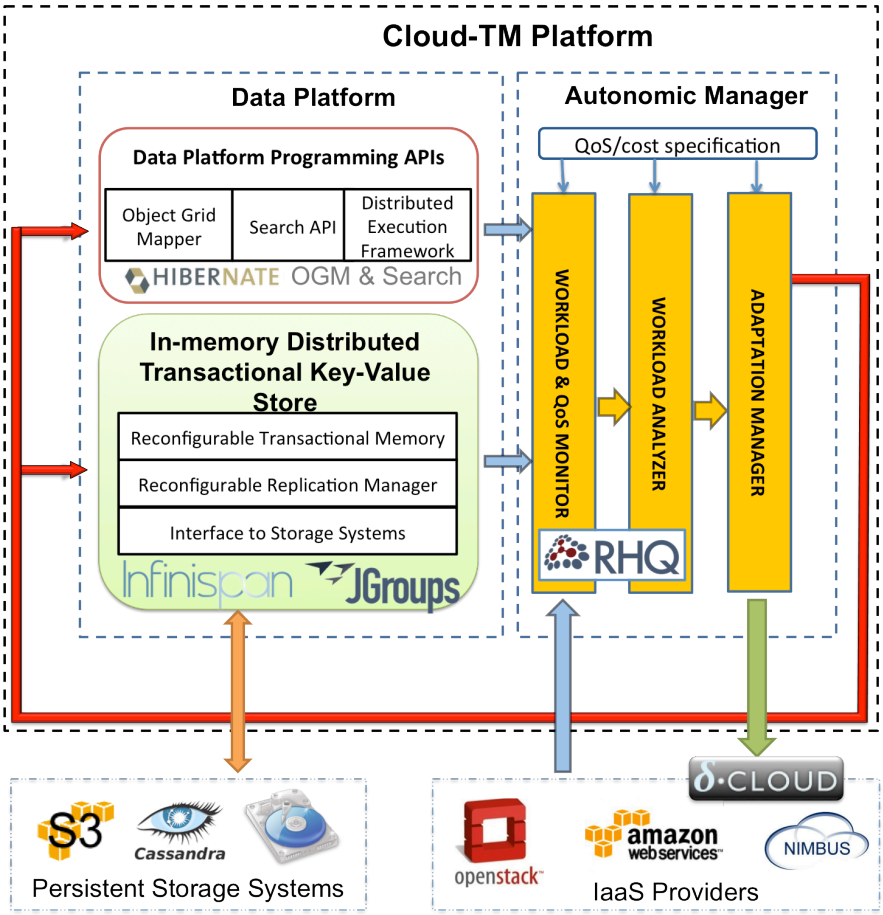

Figure 1. Architectural Overview of the Cloud-TM Platform

3. Overview of the Cloud-TM Platform

Figure 1 presents the high-level architecture of the Cloud-TM Platform, which is formed by two main parts: the Data Platform and the Autonomic Manager.

Data Platform. The Data Platform is responsible for storing, retrieving and manipulating data across a dynamic set of distributed nodes, elastically acquired from the underlying IaaS cloud provider(s).
The APIs exposed by Data Platform have been designed to simplify the development of large-scale data centric applications. To this end, they allow for:

- storing/querying data using the familiar and convenient abstractions provided by the object-oriented paradigm, such as inheritance, polymorphism, and associations;
- taking full advantage of the processing power of the Cloud-TM Platform via a set of simple abstractions that hide the complexity associated with parallel/distributed programming, such as thread synchronization, scheduling, and fault-tolerance;
- achieving strong consistency, joint to high scalability, thanks to underlying fully-decentralized multi-version distributed data management protocols, genuine partial replication techniques, and locality aware load balancing mechanisms.

Lower in the stack we find a highly scalable, adaptive in-memory transactional key-value store/Distributed Transactional Memory (DTM) [1], which represents the backbone of the Cloud-TM Data Platform. In order to maximize the visibility, impact, and future exploitation of the results of the Cloud-TM project, the consortium opted to use the Infinispan[1] open-source project by Red Hat as the starting point for developing this essential component of the Cloud-TM Platform. Infinispan has been extended with innovative data management algorithms, as well as with self-reconfiguration schemes aimed at guaranteeing optimal performance even in highly dynamic cloud environments.

Autonomic Manager. The Autonomic Manager is the component in charge of the self-tuning of the Data Platform. In Cloud-TM, self-optimization is a pervasive property that is pursued across multiple layers. Specifically, the Cloud-TM Platform leverages on a number of complementary self-tuning mechanisms that aim to automatically optimize, on the basis of user specified Quality of Service (QoS) levels and cost constraints, the following functionalities/parameters:

- the scale of the underlying platform, i.e., the number and type of machines over which the Data Platform is deployed;
- the data replication degree, i.e., the number of replicas of each datum stored in the platform;
- the protocol used to ensure transactional data consistency;
- the data placement strategies and the request distribution policies, with the ultimate goal of maximizing the data access locality.

Figure 2 illustrates an example scenario highlighting the autonomic, self-optimizing capabilities of the Cloud-TM Platform. Depending on the current workload characteristics, Cloud-TM can autonomously acquire or release cloud resources, and adjust, in a transparent manner, its internal mechanisms to maximize performance and efficiency.

[1] http://www.infinispan.org

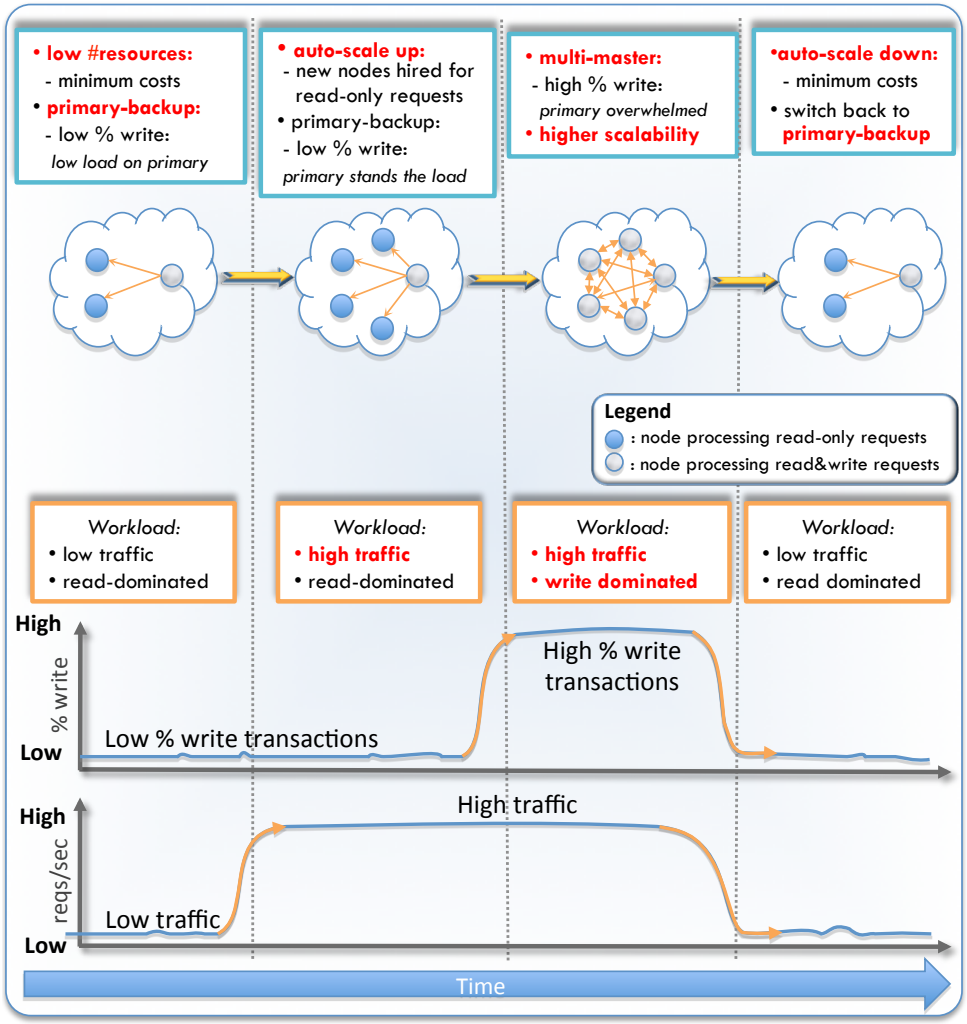

Figure 2. Self-tuning examples in the Cloud-TM Platform.

As illustrated in the diagram in Figure 1, the Autonomic Manager exposes an API allowing the specification and negotiation of QoS requirements and budget constraints.

The Autonomic Manager leverages on pervasive monitoring mechanisms that characterize the utilization of system-level resources (such as CPU, memory, network and disk), the workload and the efficiency the various subcomponents of the Data Platform (such as the local concurrency control and the data replication/distribution mechanisms).

The stream of raw data gathered by the Workload and Performance Monitor is filtered and aggregated by the Workload Analyzer, which distills workload profiling information and triggers alert signals for the Adaptation Manager.

Finally, the Adaptation Manager hosts a set of optimizers that rely on multiple techniques of different nature, ranging from analytical or simulation-based techniques to machine-learning ones. These are used to self-tune the various components of the Data Platform and to control the dynamic auto-scaling mechanism with the ultimate goal of meeting QoS/cost constraints.

3.1 The Cloud-TM In-Memory Transactional Data Grid

As hinted, the backbone of the Cloud-TM Data Platform is Red Hat's Infinispan, a highly scalable, in-memory distributed key-value store with support for transactions. The close collaboration among Red Hat teams and the academic partners of the Cloud-TM project led to integrate in Infinispan innovative data management algorithms and self-tuning mechanisms, such as the ones described in the following.

Scalable transactional consistency. The mechanisms for enforcing data consistency in presence of concurrent data manipulations and possible failures are of paramount importance for a distributed in-memory transactional data platform. In the scope of the Cloud-TM project, Infinispan was extended with a novel, scalable distributed multi-versioning scheme, called GMU [2], which has the following unique characteristics:

- **strong consistency:** GMU's consistency semantic abides by the Extended Update Serializability criterion [3], which ensures the most stringent of the standard ISO/ANSI SQL isolation levels, namely the SERIALIZABLE level. Beyond that, it ensures that the snapshot observed by transactions, including those that need to be aborted, is equivalent to the one generated by *some* serial execution of transactions. By preventing transactions from observing non-serializable states, application developers are spared from the complexity of dealing explicitly with anomalies due to concurrency, which may lead to abnormal executions [3];
- **wait-free read-only transactions:** GMU ensures that read-only transactions can be committed without the need for any validation at commit time. Further, it guarantees that read-only transactions are never blocked or aborted. These properties are extremely relevant in practice, as most real-life workloads are dominated by read-only transactions;
- **genuine partial replication:** GMU is designed to achieve high scalability also in presence of write intensive workloads by ensuring that update transactions commit by contacting exclusively the subset of nodes that maintain data they read/wrote. Hence, GMU can commit transactions without either relying on any centralized service (which is doomed to become a bottleneck) or flooding all nodes .

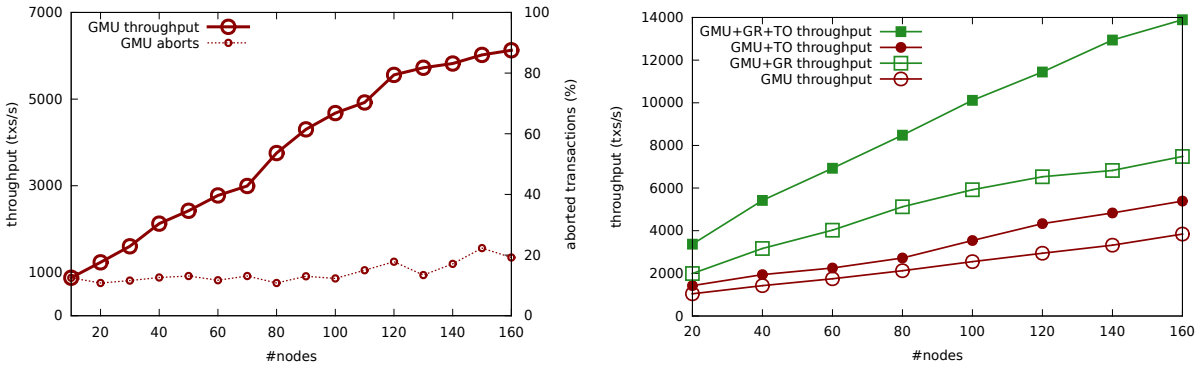

Figure 3. The GMU was shown to scale up to more than 150 nodes. The left plot uses the Vacation-Mix benchmark, whereas the right plot employs a variant of the YCSB benchmark and a version of GMU exploiting ghost reads (GR) [7] and total order multicast primitives (TO) [12].

Figure 3 provides a concise representation of scalability trends offered by the GMU protocol.

Polymorphic replication. Another unique feature of the Cloud-TM Platform is the ability to dynamically adapt its data consistency protocols, a feature that we called *polymorphic data replication* [4]. The key observation motivating this feature is that, as the vast literature in the area of transactional data consistency protocols (and our own experimental results) demonstrates, no one-size-fits-all solution exists that guarantees optimal performance across all possible workloads. In order to achieve optimal efficiency, even in presence of dynamically changing workloads, the Cloud-TM Platform:

- supports three alternative data replication strategies, obtained as variants of the GMU protocol, which exhibit different trade-offs and are, consequently, optimized for different workloads;
- allows for on-line switching between replication strategies, leveraging on innovative non-blocking schemes that minimize performance penalization during system's reconfiguration.

Self-tuning data placement. Another innovative feature developed within the context of the Cloud-TM project is the, so-called, AUTOPLACER system [5]. In a distributed data platform, processing applications' requests can require accessing remotely stored data that is stored remotely, i.e., on different nodes of the platform. Hence, application's performance and scalability can be affected significantly by the quality of the algorithms used to distribute data among the nodes of the platform. 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 deployments. AUTOPLACER addresses this problem by automatically identifying the data items having a sub-optimal placement 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

Interoperability with diverse cloud storages. Within the Cloud-TM Platform, in-memory replication represents the reference mechanism to ensure fault-tolerance without incurring in the overhead of synchronous

logging to persistent storages. Nevertheless, for durability Cloud-TM supports persistence of data towards external storage platforms via a modular, plug-in based, architecture that allows to neatly encapsulate the inherent peculiarities of the interactions with heterogeneous persistent storages via a homogeneous abstraction layer. Currently, Cloud-TM ships with plugins for the main alternative classes of persistent storage systems, such as local file-systems, DBMSs, as well as other distributed cloud storage platforms (such as Cassandra).

Extended workload characterization. Given that the In-memory Transactional Data Grid plays such a crucial role in the Cloud-TM Platform, extensive work has been done throughout the project to predict its performance (in order, e.g., to support QoS-based provisioning) and maximize its efficiency via self-tuning mechanisms. A preliminary result that we had to achieve in order to pursue these objectives was to be able to accurately (yet efficiently) trace a number of different parameters (beyond classic system resources' utilization) aimed at providing profiling information (necessary to instantiate performance models of different nature), and at identifying the presence of hot-spots (e.g., data items causing excessive contention or poor locality) and suboptimal system's configurations [6]. These statistics are valuable not only to automate the tuning/provisioning process via the Autonomic Manager, but also for human end-users, who can extract detailed information on the efficiency of their applications.

3.2 The Cloud-TM Programming API

As already mentioned, the programming model supported by the Cloud-TM Platform is designed to shelter programmers from the complexity associated with the development of large scale cloud applications. More in detail, the Cloud-TM provides APIs for both Java and Ruby, which offer the following key abstractions:

- **Object orientation**: Cloud-TM allows programmers to transparently store and manipulate object-oriented domain models, hence exposing to developers a much more expressive programming paradigm and data model than classic NoSQL key-value stores. Under the hood, the Cloud-TM Platform uses innovative object-to-key/value mapping mechanisms explicitly designed for maximizing efficiency in large scale distributed data platforms. These include support for annotations, allowing programmers to provide hints on how objects should be distributed across the platform in order to maximize locality.

- **Query support**: the Search API of the Cloud-TM Data Platform allows applications to define ad-hoc queries to retrieve and manipulate portions of the state they manage. The Search API is fully integrated with the object-oriented programming paradigm offered by Cloud-TM, supporting intrinsic aspects of the object-oriented model, such as polymorphism and inheritance. The Search API exposes to the programmer a subset of the standard Java Persistent Query Language (JP-QL) interface, as well as a new Domain Specific Language (DSL)

that supports extended features, such as advanced full-text queries, ranked searching, proximity queries, phrase queries etc. This functionality relies on an innovative design strategy that integrates some of the leading open-source projects in the area of data management and indexing, namely Hibernate ORM and Apache Lucene.

Distributed Execution Framework: The Distributed Execution Framework (DEF) provides abstractions aimed at simplifying the development of parallel applications. The DEF can be seen as a distributed version of the popular Java concurrency package, since it also provides abstractions to schedule and support the execution of parallel tasks in the platform. Unlike most other distributed frameworks, the Cloud-TM's DEF uses data maintained in local memory as input for execution tasks. This allows developing highly scalable data-centric programs, benefitting from the platform's built-in mechanisms for ensuring fault-tolerance and transactional guarantees. The DEF offers abstractions to distribute and load balance the execution of tasks while maximizing data access locality, as well as concurrency friendly distributed collections that exploit collections' semantics, and advanced concurrency mechanisms to maximize scalability in conflict-prone scenarios (like ghost reads and delayed actions [7]).

3.3 The Cloud-TM Autonomic Manager

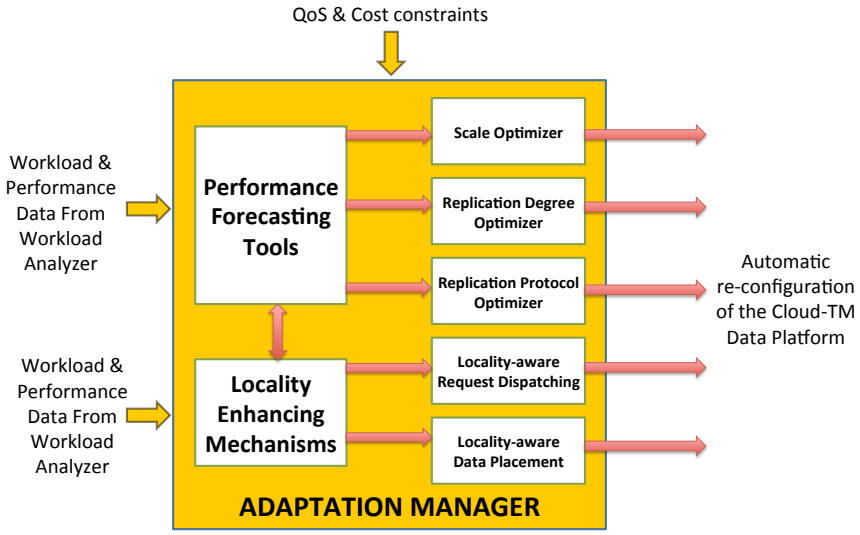

Figure 4. Self-tuning features of the Cloud-TM Autonomic Manager

The diagram of in Figure 4 highlights the set of self-tuning mechanisms supported by the Cloud-TM Autonomic Manager, which can be classified in two main types:

1. Solutions aimed at identifying the optimal values of a set of key configuration parameters/mechanisms of the Cloud-TM Data Platform, namely:

- the **scale** of the underlying platform, i.e., the number and type of nodes over which the Cloud-TM Data Platform is deployed;
- the **number of replicas** of each datum stored in the platform, which we also call, replication degree;
- the **replication protocol** to be used to synchronize the replicas' state.

The core of the self-optimization processes responsible for tuning the above parameters/mechanisms consists of a set of **performance forecasting models/tools**, which have been the subject of a number of scientific publications [10, 11, 12, 14]. These performance models are exploited to estimate the performance achievable by Cloud-TM applications when using alternative settings of the aforementioned parameters/mechanisms.

2. Mechanisms aimed at **optimizing the data access locality** of Cloud-TM applications, which strive to maximize the co-location between application code and the data it accesses. To this end, the Cloud-TM's Autonomic Manager complements the already described AUTOPLACER system with **locality-aware load distribution mechanisms**, namely request dispatching policies aimed at enhancing the locality of the data access patterns generated by Cloud-TM applications.

The actual "*brain*" of the Autonomic Manager is the **Adaptation Manager**. 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 (by transparently acquiring/releasing resources from IaaS providers). The Adaptation Manager includes two main subcomponents, the Performance Prediction Service and the Platform Optimizer.

The **Performance Prediction Service** encapsulates diverse performance forecasting mechanisms that rely on alternative predictive methodologies working in synergy to maximize the prediction accuracy and, consequently, to optimize the whole self-tuning process. Particularly, the Performance Prediction Service exploits the notion of ***model diversity***, i.e., it combines white-box (e.g., analytical) and black-box (e.g., machine-learning) approaches with complementary strengths and weaknesses in order to take the **best of each approach**, i.e.:

3. the **high accuracy** of black-box statistical methods when faced with workloads similar to those witnessed during their training phase;
4. the **minimal training** phase of white-box methods, and their high extrapolation power, i.e., their ability to achieve good accuracy even when providing forecasts concerning previously unexplored regions of the workloads' parameter space.

The Performance Prediction Service is used not only to guide the optimization process, but also to allow end-users to conduct what-if analysis aimed at assessing the performance achievable by the Cloud-TM Platform when deployed over infrastructures of different scales, and in presence of different

workloads. This type of analysis can be extremely valuable for developers of Cloud-TM applications, as well as for providers of the Cloud-TM Data Platform in PaaS environments. Developers 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. PaaS providers, on the other hand, can exploit what-if analysis, as supported by the Performance Prediction Service, to support the risk management process underlying the definition of SLAs with their customers (e.g., application providers) and to assist them in their provisioning of (physical/virtual) computation resources.

The **Platform Optimizer** 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 and extensible software architecture, which allows for configuring chains of different optimizers aimed at driving the various self-tuning schemes supported by the Cloud-TM Platform.

4. The pilots

The Cloud-TM project included two pilots, used both for demonstration and for evaluation purposes: AI-Colony, a multiplayer online game, and GeoGraph, a realistic benchmark that emulates the workload of geo-social applications. In the following, we focus on the latter.

GeoGraph has been designed in order to be flexible both in the heterogeneity and in the dynamics of the generated workloads. In particular, GeoGraph provides 19 different geo-social services and 16 application specific user simulators. These simulators can be combined in complex ways to generate dynamic workload profiles. GeoGraph's workload generator is fully distributed in order to maximize its scalability, and hence to enable the benchmarking of large-scale deployments.

GeoGraph exploits extensively the features offered by the Cloud-TM Platform. Specifically, it relies on Cloud-TM's object-grid mapping layer to transparently map an object-oriented model (implemented in Ruby) to the Cloud-TM's in-memory distributed transactional key-value store. Furthermore, GeoGraph exploits the query support provided by the Cloud-TM Data Platform to perform object lookups in an object-oriented fashion, and relies on various locality-enhancing mechanisms.

GeoGraph was used to evaluate two of the most innovative self-tuning features of the Cloud-TM Platform: AUTOPLACER [5] (i.e., the mechanism in charge of self-tuning the placement of data) and MORPHR [9] (i.e., the framework supporting the polymorphic replication abstraction).

The demonstration of AUTOPLACER (see Figure 5) is based on simulating an application, deployed over 40 nodes, in which eo-localized agents occasionally store their position and very often read posts about their surroundings,

namely a typical workload for applications like Trip Advisor. After 10 minutes of statistics gathering, AUTOPLACER starts optimizing the placement of data in the platform, yielding a considerable decrease of the number of remote operations issued over time by transactions (see right plot) and significant throughput speed-ups (above 4x).

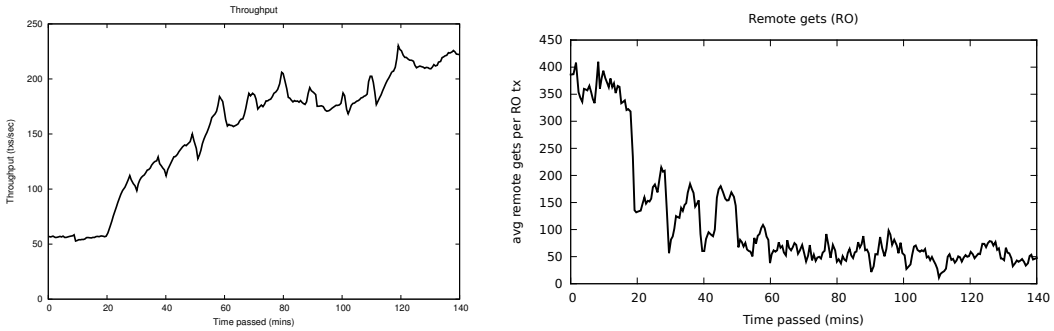

Figure 5. Demonstrating AUTOPLACER using GeoGraph

Figure 6, on the other hand, shows the benefits achievable by dynamically adjusting the replication protocol via the MORPHR framework.

In this case we configured GeoGraph to simulate a "sports aficionado" geo-social application where fans can trace their trips to sport events and generate posts about them. The simulation begins with a steady situation where users are spread across the world in 128 different locations (Phase 1). Next, in Phase 2, we simulate the start of a major event, say the Olympic opening ceremony in Rome, during which many users will share their positions, comment on the event and read posts about the surroundings. Finally, in Phase 3, the ceremony is over and the number of people who share their position starts diminishing while more users start blogging about the event.

The bar plot on the left side of Figure 6 contrasts the performance of three variants of the GMU protocol: a classic approach based on distributed locking and the two-phase commit protocol (2PC), a scheme based on total-order multicast (TO) [12], and a single-master variant in which only a single-node, the primary, is allowed to process update transactions (PB). The experimental data clearly highlights that the best performing protocol is different for each of the considered workloads, with striking differences of one order of magnitude among their relative performances.

Finally, the right plot of Figure 6 demos the MORPHR framework in action with GeoGraph. In this case we play sequentially the three above described phases (for a duration of 30 minutes each), configuring the Cloud-TM Data Platform to use PB as its initial protocol. The plots show how MORPHR can effectively detect the initially suboptimal configuration of the system and timely trigger the switch to 2PC. Further, they also show how MORPHR quickly, and correctly, reacts to further workload changes, occurring after 20 and 30 minutes since the start of the experiment, constantly ensuring the optimality of the adopted protocol's configuration.

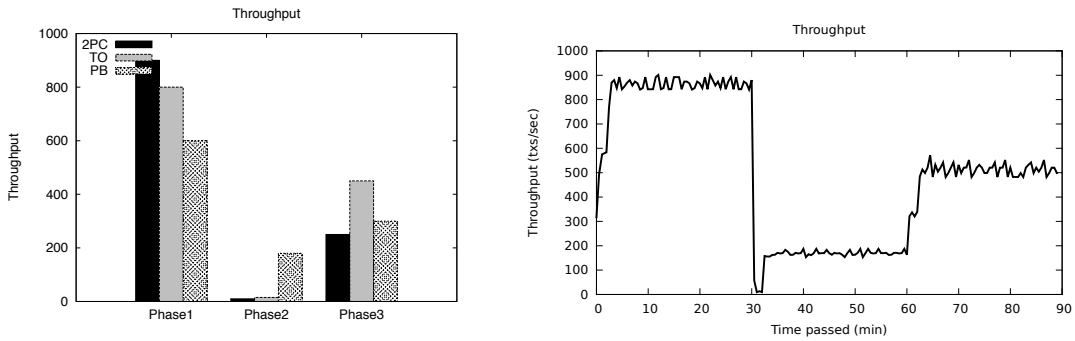

Figure 6. MORPHR in Action with GeoGraph.

The Open Source Way

Since the early stages of the project, academic partners have worked in close collaboration with the leading company in the open-source arena, Red Hat. This has allowed for integrating a number of innovative solutions into highly visible open source projects, like Infinispan, JGroups, Hibernate Search and Hibernate OGM.

The entire code of the Cloud-TM Platform has been made publicly available and released using open source licensing schemes, and is available in the GitHub repository of the project:

http://github.com/cloudtm

Additional documentation (including demo videos and **ready-to-go OpenStack-based virtual machine images**) can be downloaded from the project's webpage:

http://www.cloudtm.eu

Key lessons learnt

The Cloud-TM project has demonstrated that building highly scalable cloud applications does not imply having to embrace the complexity of weak consistency semantics (like eventual consistency) and/or low-level data models (as those offed by key-value stores). By exposing a familiar object-oriented programming paradigm, enriched with the intuitive abstraction of atomic transaction, Cloud-TM hides most of the sources of complexities associated with building data-centric cloud applications, while being able to scale out up to hundreds of machines. Clearly achieving this goal was far from being trivial, and below we report the key lessons that we learnt while building the Cloud-TM Platform:

- ***Maximizing data locality***: popular data placement schemes based on consistent hashing tend to perform quite poorly with object-oriented transactional applications, in which the processing of applications' requests frequently involves accessing a large number of

data items. In Cloud-TM we tackled this problem by combining dedicated programming constructs, which enable programmers to convey domain-specific information on which objects are likely to be accessed in the same transaction, with machine-learning techniques, which identify at run-time optimized data placement strategies based on the access patterns generated by applications.

- ***Minimizing data conflicts***: in large scale systems logical conflicts among transactions can quickly become a major scalability constrainer. The Cloud-TM Platform copes with this issue in a twofold way: providing programming abstractions (like dirty reads or delayed actions [7]) that empower developers to minimize data contention and optimize the management of critical contention hot spots; self-tuning the concurrency control/replication mechanisms employed by the data platform to best match applications' workload characteristics [4,9].
- ***Genuine partial replication and efficient read-only transactions***: most realistic workloads tend to be read-dominated, making it important to optimize the processing of read-only transactions. Multi-versioning concurrency control (MVCC) is universally recognized as the "way-to-go" to optimize performance of read-only transactions, as MVCC can spare them from aborting or blocking. Unfortunately, designing scalable *distributed* MVCC algorithms is quite challenging, as, in distributed settings, it is hard to totally order any new data item versions in a scalable way, namely avoiding centralized components or involving all nodes in the commit of every update transaction (which is the so called non-genuine partial replication approach). In Cloud-TM we addressed this problem by introducing GMU, the first protocol to support waits-free read-only transactions and genuine partial replication, while preserving ease of programming by ensuring consistency semantics very close to 1-Copy Serializability.

Another key lesson learnt during this project is that the performance prediction of cloud data platforms like Cloud-TM severely challenges both conventional analytical/white-box modeling techniques (e.g., based on queuing theory) as well as more recent approaches based on machine-learning/black-box methodologies. Indeed, building accurate white-box models of these types of systems is a major conundrum given their inherent complexity and the lack of detailed information on the actual underlying hardware infrastructure/topology - which is the norm for virtualized cloud infrastructures; black-box models, on the other hand, fall quickly prey of the curse of dimensionality, typically requiring prohibitive training times to gather knowledge bases large enough to convey representative information on the performance of the system in a wide spectrum of workloads and deployments/configurations (e.g., large vs small scale deployments, full vs partial replication scenarios, public vs private cloud deployments).

In Cloud-TM we tackled this challenge by exploring new ways of combining black-box and white-box modeling, in order to achieve the best of the worlds. Although the concept of gray-box modeling is not new, the research on applying such techniques for the performance prediction/self-tuning of cloud data stores is still in its infancy, and we are not aware of any systematic study

aimed at assessing pros and cons of existing grey-box methodologies in this emerging, yet so relevant domain. This is one of the several open research questions in the area of self-tuning of cloud-data platforms, which we intend to further address in our future work.

Acknowledgments

The Cloud-TM project (257784) started on June 2010 and ended on September 2013. The project's consortium was coordinated by INESC-ID (Portugal), and included Algorithmica (Italy), CINI (Italy), and Red Hat (IE).

Biographical sketches

Paolo Romano is an assistant professor at Instituto Superior Tecnico, Universidade de Lisboa and a senior researcher at INESC-ID. His research interests include cloud computing, autonomic computing, parallel and distributed computing. He is a member of IEEE and ACM. Email: romano@inesc-id.pt

Mark Little is a VP at Red Hat where he leads JBoss technical direction, research and development. He has worked in the area of reliable distributed systems since the mid-80s. His PhD was on fault-tolerant distributed systems, replication and transactions. He is currently also a professor at Newcastle University. Email: mlittle@redhat.com

Francesco Quaglia is an associate professor at Sapienza University of Rome. His research interests are in distributed systems, middleware platforms, parallel computing applications and fault-tolerant programming. In these areas, he has authored or coauthored more than 130 technical articles. Email: quaglia@dis.uniroma1.it

Luis Rodrigues is a professor at Instituto Superior Tecnico, Universidade de Lisboa and a researcher at INESC-ID where he now servers in the board of directors. His research interests lie in the area of reliable distributed systems. He is co-author of more than 150 papers and 3 textbooks on these topics.

Vittorio Amos Ziparo is founder and director of R&D at Algorithmica Srl. He obtained a PhD from Rome University La Sapienza in multi-robot systems. His research interests include artificial intelligence, robotics and multi-agent systems. Email: ziparo@algorithmica.it

References

[1] M. Herlihy, Y. Sun. "Distributed transactional memory for metric-space networks." Distributed Computing., 2005.

[2] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, L. Rodrigues, "When Scalability Meets Consistency: Genuine Multiversion Update Serializable Partial Data Replication", Proc. IEEE ICDCS, 2012

[3] A. Adya, “Weak Consistency: A Generalized Theory and Optimistic Implemen tations for Distributed Transactions,” PhD Thesis, MIT, 1999.

[4] M. Couceiro, P. Romano, L. Rodrigues, "PolyCert: Polymorphic Self-Optimizing Replication for In-Memory Transactional Grids", Proc. ACM/IFIP/USENIX Middleware 2011

[5] J. Paiva, P. Ruivo, P. Romano, L. Rodrigues, "AutoPlacer: scalable self-tuning data placement in distributed key-value stores", Proc. USENIX ICAC, 2013

[6] D. Didona, P. Di Sanzo, R. Palmieri, S. Peluso, F. Quaglia, P. Romano, "Automated Workload Characterization in Cloud-based Transactional Data Grids", Proc. IEEE DPDNS, 2012

[7] N. Diegues and P. Romano, "Bumper: Sheltering Transactions from Conflicts", Proc. IEEE SRDS, 2013

[8] M. Couceiro, P. Romano, L. Rodrigues, "A Machine Learning Approach to Performance Prediction of Total Order Broadcast Protocols", Proc. IEEE SASO, 2010

[9] M. Couceiro, P. Ruivo, P. Romano, L. Rodrigues, "Chasing the Optimum in Replicated In-memory Transactional Platforms via Protocol Adaptation", Proc. IEEE/IFIP DSN, 2013

[10] D. Didona, P. Romano, S. Peluso, F. Quaglia, "Transactional Auto Scaler: Elastic Scaling of In-Memory Transactional Data Grids", Proc. USENIX ICAC, 2012

[11] 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" , Proc. SIMUTools, 2013

[12] P. Ruivo, M. Couceiro, P. Romano, L. Rodrigues, "Exploiting Total Order Multicast in Weakly Consistent Transactional Caches", Proc. PRDC, 2011