Abstract
Cloud-TM is a highly innovative data-centric middleware platform, aimed at facilitating 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 modelling, 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 both:

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. low vs high number of nodes, local vs geographical distribution) on which these mechanisms are deployed.

The complexity of this problem is hence particularly exacerbated in Cloud Computing platforms due to the feature that is regarded as one of the key advantages of the cloud: its ability to elastically acquire or release resources, de facto dynamically varying the scale of the platform in real-time to meet the demands of varying workloads.

### 2. The Cloud-TM Approach

The Cloud-TM project addressed precisely the above described issues by building a **highly innovative data-centric middleware platform**. The Cloud-TM platform is designed from the grounds up to meet the scalability and dynamicity requirements of cloud infrastructures, 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 computing infrastructures embrace weak consistency models that achieve scalability at the cost 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 and allowing ordinary programmers to unleash the potentiality of large-scale Cloud platforms. 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 antagonists, 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 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 the efficiency** (i.e. the costs/benefits ratio in 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.

This means making the most effective use of the currently allocated resources, leading to a reduction in resource utilization, and, consequently, of the operational costs.

Figure 1. Architectural Overview of the Cloud-TM platform
3. Overview of the Cloud-TM Platform

Figure 1 presents the high level architecture diagram of the Cloud-TM Platform. The Cloud-TM Platform 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 the Cloud-TM Data Platform have been designed to simplify the development of large scale data centric applications deployed on cloud infrastructures. To this end, the programmatic interfaces offered by the Cloud-TM Data Platform allow to:

- store/query data into/from the Data Platform using the familiar and convenient abstractions provided by the object-oriented paradigm, such as inheritance, polymorphism, and associations.
- take 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.
- enable the joint achievement of high scalability and strong consistency via fully-decentralized multiversioning 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 [Her05,Rom08] (DTM), 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 Red Hat's Infinispan\(^1\) 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 real-time 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 the Cloud-TM Platform, 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. number of replicas of each datum stored in the platform.

\(^1\) http://www.infinispan.org
• the protocol used to ensure transactional data consistency.
• the data placement strategies and request distribution policies, with the ultimate goal of maximizing the data access locality of Cloud-TM applications.

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.

As illustrated in the diagram of Figure 1, at its topmost layer, 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 both 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 (local concurrency control, data replication and distribution mechanisms, data contention level).

The stream of raw data gathered by the Workload and Performance Monitor component is filtered and aggregated by the Workload Analyzer, which generates distilled workload profiling information and triggers alert signals that serve as input 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 models to machine-learning-based mechanisms. These techniques self-tune the various components of the Data Platform and 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 a highly scalable, in-memory distributed key-value store with support for transactions. Infinispan, an open-source project sponsored by Red Hat, has been selected as the reference distributed transactional platform for the project. The close collaboration among Red Hat teams and the academic partners of the Cloud-TM project led to the integration in Infinispan of highly innovative data management algorithms and self-tuning mechanisms, such as:

Highly-scalable transactional consistency schemes. 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 project, Infinispan has been extended with a novel, highly scalable distributed multi-versioning scheme, called GMU [Pel12], which has the following unique characteristics:

- **strong consistency:** GMU’s consistency semantics abides by the Extended Update Serializability criterion [Ady99], 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, even 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 that may lead to abnormal executions [Gue08].

- **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 update transactions without either relying on any centralized service (which is doomed to become a bottleneck) or flooding all nodes (saturating the network).

![Graph showing GMU throughput and aborts](image1)

**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) [Die13] and total order multicast primitives (TO) [Rui11].

**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* [Cou11]. The key observation motivating this feature is that, as the vast literature in the area of transactional data consistency protocols (as well as our own experimental results) demonstrate, 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, which exhibit different trade-offs and are, consequently, optimized for different workloads;
- allows for on-line switching between replication protocols, leveraging innovative non-blocking schemes that minimize performance penalization during system’s reconfiguration.

**Self-tuning Data Placement.** Another highly innovative feature developed within the context of the Cloud-TM project is the, so called, **AUTOPLACER** scheme [Pai13]. 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 Cloud-TM’s 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 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.

**Interoperability with diverse cloud storages (and not only).** Within the Cloud-TM Platform, in-memory data replication represents the reference mechanism to ensure fault-tolerance and data reliability without incurring in the overheads of synchronous logging to persistent storages. Nevertheless, in order to ensure data 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 underlying 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 Distributed 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 objective was to be able to accurately (yet efficiently) trace a number of different metrics (beyond classic system resources' utilization) aimed to provide profiling information (required to instantiate performance models of different nature), identify the presence of hot-spots (e.g., data items causing excessive contention or poor locality) and suboptimal system's configurations [Did12-a]. These statistics are valuable not only to automate the tuning/provisioning process via the Autonomic Manager, but also for human end-users, who can exploit them to extract detailed profiling information on the efficiency of their applications.

**3.2 The Cloud-TM Programming API**

As already mentioned the programming model of the Cloud-TM Platform is designed to shelter programmers from the complexity associated with the development of applications to be deployed on large scale distributed cloud infrastructures. More in detail, the Cloud-TM Platform 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-grid 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 of 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 is structured in two new components based 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 to simplify the development of parallel applications. The DEF can be seen as a distributed version of the popular Java concurrency package, as it also provides abstractions to schedule and support the execution of parallel tasks in the platform. Unlike most other distributed frameworks, the Cloud-TM Distributed Execution Framework 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 (such as the Locality-Aware Request Dispatching), as well as concurrency friendly distributed collections (such as Linked Lists and B+ trees) that exploit collections' semantics, and advanced concurrency mechanisms to maximize scalability in conflict-prone scenarios [Die13].
3.3 Autonomic Manager

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

- 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;
  - **number of replicas** of each datum stored in the platform, which we also call, replication degree;
  - **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 [Cou10, Cou13, Did12, Dis12, Dis13]. These performance models are leveraged to estimate the performance achievable by Cloud-TM applications when using alternative settings of the above mentioned parameters/mechanisms, with the ultimate goal of maximizing their efficiency.

- Mechanisms aimed at **optimizing the data access locality** of Cloud-TM applications, that is targeted at maximizing the co-location between the application code and the data it accesses, with the main objective to maximize efficiency and scalability. To this end two main mechanisms have been integrated in the Cloud-TM's AM:
mechanisms aimed at automatically identifying the optimal placement of data replicas across the nodes of the Cloud-TM Platform, i.e. the allocation of data replicas to nodes that minimizes the number of remote accesses generated by Cloud-TM applications. The optimization keeps into account constraints on the capacity of the nodes and on the minimum/maximum number of copies maintained in the system;

- mechanisms aimed at automatically defining locality-aware load distribution policies, that is request dispatching policies capable of 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 accuracy of the prediction system, and, consequently, of the whole self-tuning process. In more detail, the Performance Prediction Service 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, i.e.:

- 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 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 detailed 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, as well as for providers of the Cloud-TM Data Platform in a PaaS environment. 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.
(i.e., 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 includes two pilots used both for demonstration and 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, we provide 19 different geo-social services (i.e., actions) 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 enable the benchmarking of large scale deployments of the Cloud-TM Platform.

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 supports provided by the Cloud-TM Data Platform to perform object lookups in an object-oriented fashion, and relies on various locality enhancing mechanisms.

GeoGraph has been chosen to evaluate and demonstrate two of the most innovative self-tuning features of the Cloud-TM Platform: **AUTOPLACER** [Pai13] (i.e., the mechanism in charge of self-tuning the placement of data) and **MORPHR** [Cou13] (i.e., the framework supporting the dynamic adaptation of the replication schemes).

The demonstration of **AUTOPLACER** is based on a simulation that emulates an application where agents occasionally store their position and very often read posts about their surroundings, i.e. a typical workload for applications like Trip Advisor. Figure 5 shows the results of using AUTOPLACER. AUTOPLACER is triggered after around 10 minutes since the beginning of the test. After 10 minutes of statistics gathering, AUTOPLACER starts optimizing the placement of data in the platform: the result is a considerable decrease of the number of remote operations issued over time by transactions (see right plot), which yields significant throughput speed-ups (above 4x).
Figure 5. Demonstrating AUTOPLACER using GeoGraph.

Figure 6. Performance of different replication protocols when faced with different workloads.

Figure 7 demonstrates 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 7. Demonstrating MORPHR using GeoGraph.](image)

**Open Source**

Since the early stages of the project, academic partners have worked in close collaboration with the leading company in the open-source software arena, Red Hat. This has allowed to integrate a number of innovative solutions in 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 (typically LGPL, with some components released under Apache 3 and GPL licenses).

The **source code of the Cloud-TM Platform** is available in the GitHub repository of the project:

http://github.com/cloudtm

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

http://www.cloudtm.eu

**Acknowledgments**

Tha Cloud-TM project (Grant no. 257784, FP7) started on the 1st of June 2010 and ended on the 1st of September 2013. The project’s consortium was coordinated by Inesc-ID (Portugal), and included Algorithmica S.r.l. (Italy), CINI (Italy), and Red Hat (IE).
References


