# Automated Workload Characterization in Cloud-based Transactional Data Grids

Diego Didona*, Pierangelo Di Sanzo †, Roberto Palmieri†,
Sebastiano Peluso†, Francesco Quaglia†, and Paolo Romano*
*INESC-ID/IST, Lisbon, Portugal
†Sapienza Rome University, Rome, Italy

***Abstract*—Cloud computing represents a cost-effective paradigm to deploy a wide class of large-scale distributed applications, for which the pay-per-use model combined with automatic resource provisioning promise to reduce the cost of dependability and scalability. However, a key challenge to be addressed to materialize the advantages promised by Cloud computing is the design of effective auto-scaling and self-tuning mechanisms capable of ensuring pre-determined QoS levels at minimum cost in face of changing workload conditions.**

**This is one of the keys goals that are being pursued by the Cloud-TM project, a recent EU project that is developing a novel, self-optimizing transactional data platform for the cloud. In this paper we present the key design choices underlying the development of Cloud-TM's Workload Analyzer (WA), a crucial component of the Cloud-TM platform that is change of three key functionalities: aggregating, filtering and correlating the streams of statistical data gathered from the various nodes of the Cloud-TM platform; building detailed workload profiles of applications deployed on the Cloud-TM platform, characterizing their present and future demands in terms of both logical (i.e. data) and physical (e.g. hardware-related) resources; triggering alerts in presence of violations (or risks of future violations) of pre-determined SLAs.**

## I. Introduction

The Cloud Computing paradigm is profoundly changing both the architectures of the IT systems and the organization of the enterprise IT infrastructure management. Architectures of distributed computing platforms are moving from a traditional static model, where the amount of resources allocated to applications/services are a-priori estimated, towards an elastic model, where resources can be provisioned on-demand. The flexibility of Cloud computing's pay-per-use model is driving many enterprises to move their IT infrastructure and services to the "cloud". Anyway this new paradigm opens up new challenges.

One of these is related to the design of effective auto-scaling and self-tuning mechanisms capable of ensuring pre-determined QoS levels at minimum costs in face of changing workload conditions. In fact, on one hand, an elastic platform provides a very cost-effective model to improve system performance and dependability by means of data and services replication. On the other hand, in order to take advantage of the elasticity of the underlying infrastructure, both the data and services replication management need to be automated by means of mechanisms able to guarantee the desirable Quality of Service (QoS) levels while minimizing the operational cost of the infrastructure. In such a context, the efficiency of the adopted replication scheme plays a role of paramount importance. In fact, in environments characterized by dynamic and/or heterogeneous workloads, the costs due to the replication can be highly variable and, as a consequence, the dynamic configuration and/or selection of the replication mechanisms represent crucial aspect to reduce the cost of dependability.

This is one of the key goals that is being pursued by the Cloud-TM project [1], a European Project that is developing a distributed self-optimizing data management platform tailored for cloud environments. The Cloud-TM platform includes auto-tuning policies aimed at automating the system (re-)configuration in order to meet user-specified SLAs (in terms of both performance and fault-tolerance) at minimum operational costs. This is achieved via the joint-usage of elastic scaling schemes, aimed at automating provisioning of infrastructural (e.g. computational or storage) resources, and pervasive self-optimization strategies that dynamically select the optimal replication algorithm to use, given the scale of the platform and the workload characteristics of the application deployed on it.

In this paper we present the key design choices underlying the development of Cloud-TM's Workload Analyzer (WA), a crucial component of the Cloud-TM platform that is in charge of gathering and extracting the information on whose basis the self-tuning policies, driving the auto-configuration process of the platform, can be defined. More specifically, the WA provides three key functionalities: i) aggregating, filtering and correlating the streams of statistical data gathered from the various nodes of the Cloud-TM platform; ii) building detailed workload profiles of applications deployed on the Cloud-TM platform, characterizing their present and future demands in terms of both logical (i.e. data) and physical (e.g. hardware-related) resources; iii) triggering alerts in presence of violations (or risks of future violations) of pre-determined SLAs.

The rich set of functionalities provided by Cloud-TM's WA is obtained by combining, on one side, leading open source projects in the areas of systems management [2], statistical computing [3] and stream-based algorithms [4], and on the other, innovative methodologies for what concerns the characterization of transactional applications [5].

The paper presents also the results of an experimental study highlighting the efficiency of the proposed solution that allows extracting detailed workload information at the cost of

a negligible overhead.

The remainder of this paper comprises three sections. In Section II an overview of the Cloud-TM platform is provided. The WA is presented in Section III, where we describe its key functionalities, its main components and the evaluation study. Finally, Section IV concludes the paper.

## II. Overview of the Cloud-TM Platform

Figure 1 overviews the architecture of the Cloud-TM platform. It encompasses two main logical components: the Data Platform and the Autonomic Manager.

The Data Platform is responsible for retrieving, manipulating, and storing data across a dynamic set of distributed nodes that are elastically acquired from one or more underlying IaaS Cloud providers. It will expose a set of APIs, denoted as Data Platform Programming APIs in Figure 1, aimed at increasing the productivity of Cloud programmers in two ways:

- By allowing ordinary programmers to read/write data from/to the Data Platform using the familiar abstractions provided by the object-oriented paradigm, such as inheritance, polymorphism, associations.
- By allowing ordinary programmers to take full advantage of the processing power of the Cloud-TM platform via a set of abstractions that will hide the complexity associated with parallel/distributed programming, such as load balancing, thread synchronization, scheduling, or fault-tolerance.

The main component of the Data Platform, is a highly scalable, elastic and dynamically Reconfigurable Distributed Software Transactional Memory (RDSTM). As starting point for developing this essential component of the Cloud-TM platform, it has been selected to use Red Hat's Infinispan [6], a recent in-memory transactional data grid designed from the ground up to be extremely scalable. Infinispan is being extended with new algorithms both for data replication and distribution, and real-time self-tuning schemes aimed at guaranteeing optimal performance even in highly dynamic Cloud environments.

The lowest level of the Data Platform provides abstractions that allow state to be persisted over a wide range of heterogeneous durable storage systems, from local/distributed filesystems to Cloud storages. The Autonomic Manager is the component in charge of automating the elastic scaling of the Data Platform, as well as of orchestrating the self-optimizing strategies that will dynamically reconfigure the data distribution and replication mechanisms to maximize efficiency in scenarios entailing dynamic workload. Its topmost layer will expose an API allowing the specification and negotiation of QoS requirements and budget constraints. The Autonomic Manager will collect information not only about heterogeneous system-level resources (such as CPU, memory, network and disk), but will also characterize the workload of each of the components of the Data Platform and their efficiency.

To this end, the streams of statistical data collected by the Workload & QoS Monitor are aggregated, filtered and correlated by the Workload Analyzer, triggering whether needed alerts (e.g. in presence of violations, or risks of violations, of pre-agreed SLAs) towards the Adaptation Manager. The Adaptation Manager is responsible for self-tuning the various components of the Data Platform and control the dynamic auto-scaling mechanism with the ultimate goal of enforcing predetermined QoS levels while minimizing the operational costs of the system.

## III. Cloud-TM's Workload Analyzer

As showed in Figure 1 the Workload Analyzer (WA) is the component in between the Workload and Performance Monitor (WPM) and the Adaptation Manager (AM). The WPM is a flexible, scalable and easy to (re-)configure subsystem responsible of audit data for both infrastructure resources and platform (or application) level components in an integrated manner [7]. The AM is in charge of determining the adaptation criteria according to either i) on-line computed performance forecasts, or ii) off-line computed adaptation policies (e.g. based on thresholds).

The WA bears the following responsibilities in the Cloud-TM platform (see Figure 2):

- **Data aggregation and filtering**: 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/filtering statistics originated by different software layers and/or groups of nodes.
- **Workload and resource demand characterization**: the WA allows for deriving detailed transactional profiles that include a number of statistical information characterizing the resource usage demand of applications deployed in the Cloud-TM platform both at the physical (e.g. CPU, memory, etc.) and data (e.g. probability of conflicts among transactions, identification of hot spots for lock contention and remote reads) levels.
- **Workload and resource demand prediction**: the WA integrates a set of scripts/interfaces allowing for using the ample library of statistical functionalities implemented by the $R$ [3] free software project. This opens the possibility to run a wide range of time-series analysis methods (such as, moving averages, ARIMAX models, Kalman filters [8]) aimed to forecast future trends of the workload fluctuations.
- **QoS monitoring and alert notification**: the WA allows for graphing raw or aggregated statistics (e.g. on the performance or availability of some servers/services), and defining complex alert conditions on the base of the collected data.

A key decision taken within the Cloud-TM consortium was to build the WA by capitalizing on several existing open-source software packages, which currently represent leading solutions in their application domain:

- **RHQ**: The RHQ project [2] is a popular systems management suite that provides extensible and integrated systems

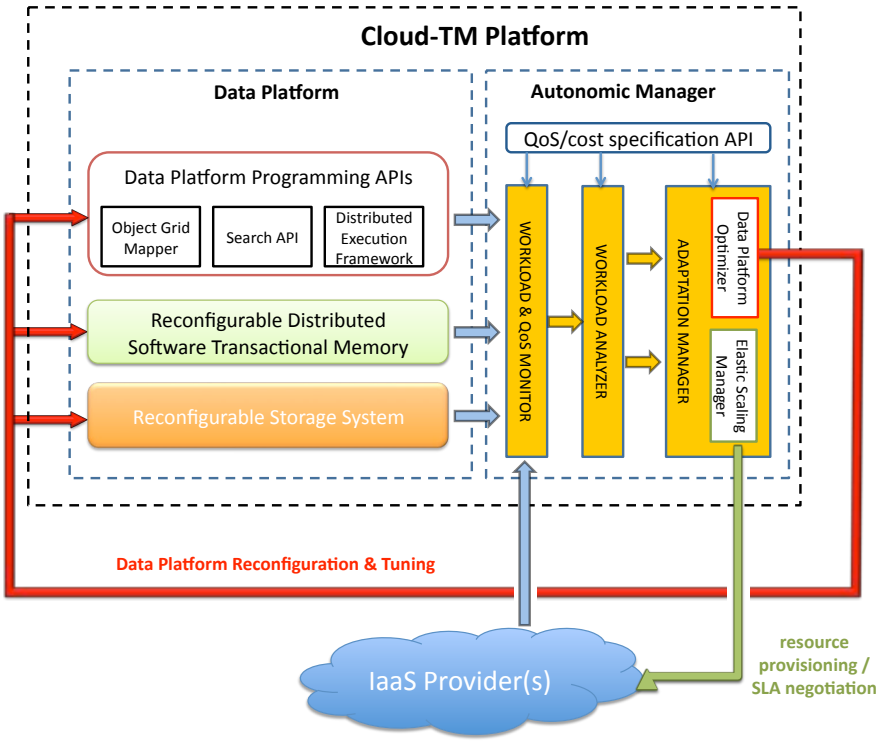


Fig. 1. Architectural Overview of the Cloud-TM Platform.

management for multiple products and platforms. Developed as an open source by Red Hat, RHQ is designed with layered modules that provide a flexible architecture for deployment. It delivers a core user interface that provides audited and historical management across an entire enterprise. A Server/Agent architecture provides remote management and plugins implement all specific support for managed products.

The RHQ project provides industrial-quality implementations of some of the key functionalities required by the WA (and more in general by Cloud-TM's Autonomic Manager, see Figure 1), including monitoring and graphing of values, alerting on error conditions or whenever a particular events occur, remote configuration of managed resources, remote operation execution, provisioning of software onto managed machines.

- **Stream-lib**: stream-lib is an open-source JAVA library that integrates a number of recent algorithms for summarizing data in streams on-the-fly, namely avoiding to store all events in the stream [9]. As we will detail in Section III-B, we use stream-lib in order to identify, using lightweight probabilistic top-k algorithms, hot spots of different nature in the data access patterns generated by transactions running in the Cloud-TM platform.
- **R-project**: $R$ is a language and environment for statistical computing and graphics, which provides a wide variety of statistical, including linear and non-linear modelling, classical statistical tests, time-series analysis, classification and clustering. As we will detail in Section III-C, the WA exploits the ample library of statistical functions provided by $R$ in order to derive workload forecasts using a range of time series analysis methodologies (ranging from simple moving averages, to more complex ARMA models and Kalman filter-based predictors), allowing for identifying trends and seasonal components among the collected data.

The diagram in Figure 2 depicts the architecture of the WA. Let us analyze it more in detail, discussing how the above mentioned open-source projects have been extended and integrated in the Cloud-TM platform architecture.

A first important step has been to extend the set of statistics exported by the components of the Cloud-TM Data Platform in order to generate a detailed profile of the transactional workload in input to the system. More details on this can be found in Section III-B.

The next step has been integrating the WPM framework with the RHQ infrastructure. This was achieved by developing an ad-hoc RHQ plug-in, designed for being fully compatible with the WPM's Log Service component (LS) output [7]. The plug-in externalizes to RHQ the statistics collected by the Cloud-TM nodes that are being monitored by the WPM. In order to decouple the LS from its RHQ plug-in (e.g. allowing to deploy them on different machines), the plug-in registers a set of listeners on an Infinispan cache, which is populated by the LS whenever a new sample (or batch of samples) is

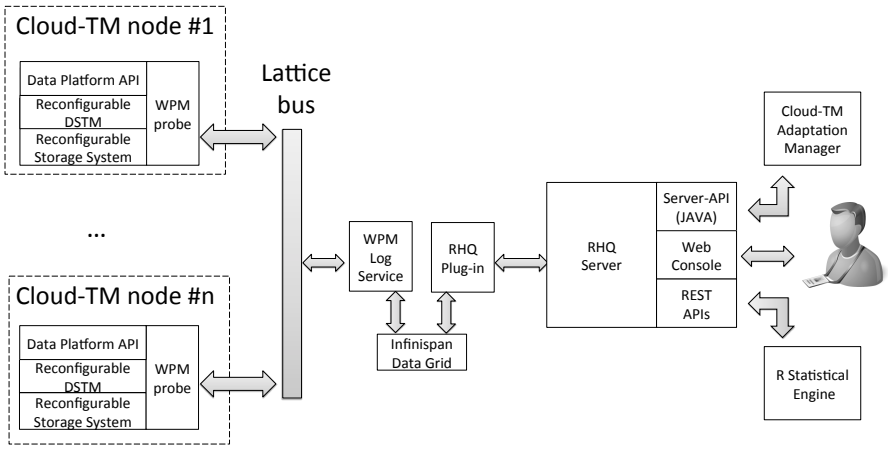


Fig. 2. Architectural Scheme of the Workload Analyzer.

gathered from the monitored nodes. This way, we exploit the fault-tolerance features of Infinispan in order to ensure high availability of the communication bus between LS and its RHQ plug-in.

Finally, once the monitoring data is convoyed by RHQ, we use its rich set of APIs and interfaces as building blocks to support a breadth of functionalities for workload analysis and forecasting.

### A. Data aggregation and filtering

In order to support aggregation and filtering of incoming monitoring data streams, the WA exploits the advanced grouping functionalities provided by RHQ. In the latter, groups serve a twofold purposes:

- defining which access permission are applied to resources monitored by RHQ inventory;
- providing a way to view aggregate data and perform actions across all group members en mass.

RHQ enables flexible group membership policies, which support not only the manual addition of resources to groups, but also the definition of regular expressions, called DynaGroups, that maintain groups membership in a dynamic fashion.

Once groups are defined, it is possible to specify control access polices directly to groups of resources, instead of individual resources. By using DynaGroups, one can effectively create dynamic ACLs (access control lists) to lessen the burden of security maintenance, especially against large inventories

Compatible groups (those composed entirely of the same type of resource, e.g. all Linux platforms, all JBossAS servers) have additional features available to them, such as: group-wise availability; min, max, and average metrics across the group; aggregate events viewer; operations against all group members, either serialized or concurrent execution policies; fine-grained changes to connection properties and resource configuration across one or more members of the group.

### B. Workload and resource demand characterization

In order to characterize the workload and resource demand of transactional applications deployed on the Cloud-TM platform, the workload analyzer acquires a large amount of statistical information from the various layers of the Cloud-TM platform. We focus in this paper on discussing the main statistical information that we employed to characterize the transactional profile of Cloud-TM applications. These statistics are collected using a number of probes scattered across several sub-components of the Infinispan data grid (e.g. Lock Manager, Distribution Manager, Rpc Manager) and are externalized using JMX interfaces [10] in order to permit their monitoring via standard JMX-based consoles or applications (and by the WPM).

We classify the statistics into *high-level* and *low-level* statistics, and describe them in the following.

*1) High Level Statistics:* High-level statistics can be in their turn distinguished in two classes:

- statistics that identify hot spots data items for two essential subsystems of a data grid: the data placement and concurrency management schemes;
- statistics aimed at identifying the maximum degree of data parallelism for an application.

For what concerns hot-spots identification, we trace the top-k keys (where k is a parameter that is dynamically configurable via JMX) that have been:

i) updated (using the put command);
ii) either remotely or locally read - thus requiring or not a remote interaction with another node during transaction execution;
iii) locked, causing no contention, contention, or abort, of a transaction.

This information is extremely valuable for the automatic and human-driven tuning of these performance-critical modules of the system, and we plan to make use of this info into the Autonomic Manager component, in order to develop different kinds of self-optimizing strategies.

In order to minimize overheads, we identify these keys using recently proposed top-k algorithms for data stream analysis. In particular we used the algorithm presented in [9] (implemented by the stream-lib opensource project [4]): this algorithm gives up the goal of providing exact guarantees, but analyze streams using a limited (constant) memory space, thus optimizing performance and lending itself to the analysis of massive streams of data.

An other key high level statistic computed by the Workload Analyzer is an innovative metric, which we named Application Contention Factor (see the technical report [5] for more details on it), that allows for characterizing the maximum degree of data parallelism exhibited by transactional applications.

In order to explain more rigorously its definition, it is required to introduce some background concepts at the basis of the analytical performance modelling approaches of transactional systems presented so far in literature. Existing works in this area [11], [12] share a common reliance on queuing theoretical arguments to derive the transaction contention probability. Denoting with $\lambda$ the average arrival rate of locks to a data item, and assuming that locks are held for an average time $T_H$, one can model a data item as a queue and approximate the probability of encountering lock contention on a data item with the utilization of the corresponding queue (namely, the fraction of time during which the data item is locked), which is computable as [13]: $U = \lambda_{lock} T_H$ , assuming $\lambda_{lock} T_H < 1$. Then, assuming that accesses are uniformly distributed on one [12] (or more [14]) set of data items of cardinality $D$, a-priori known, it is possible to compute the probability of lock contention on any of the data items simply as:

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

Unfortunately, the availability of information on $D$, and the assumption on the uniformity of the data access patterns strongly limits the employment of these models with complex applications, especially if these exhibit dynamic shifts in the data access distributions.

The idea underlying the definition of the Application Contention Factor (ACF) is to extract the equivalent value of $D$ for an application in execution on the Cloud-TM platform by exploiting the availability of information on $P_{lock}, \lambda_{lock}$ and $T_H$ in the current configuration. Given $P_{lock}, \lambda_{lock}$ and $T_H$, in fact, we can invert Eq. 1 and obtain the Application Contention Factor (ACF) as:

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

By equation 1, it follows that $\frac{1}{ACF}$ can be interpreted as the size $D$ of an "equivalent" set $\mathcal{DB}$ of data items, such that, if the application issues lock requests on disjoint data items selected uniformly from set $\mathcal{DB}$, it would incur in the same contention probability that it experienced in the current configuration.

From an other perspective, the ACF (or better, its inverse) represents the maximum number of transactions that can be concurrently executed in the system assuming that each transaction holds its locks for a single time unit. The ACF allows for characterizing the application data access pattern distribution in a very concise, lightweight and pragmatical manner, abstracting over arbitrarily complex data access patterns (e.g. with strong skew or complex analytical representation) and over the effects of contention on physical resources (abstracted away by normalizing the ACF with respect to $T_H$) via an easily tractable analytical model.

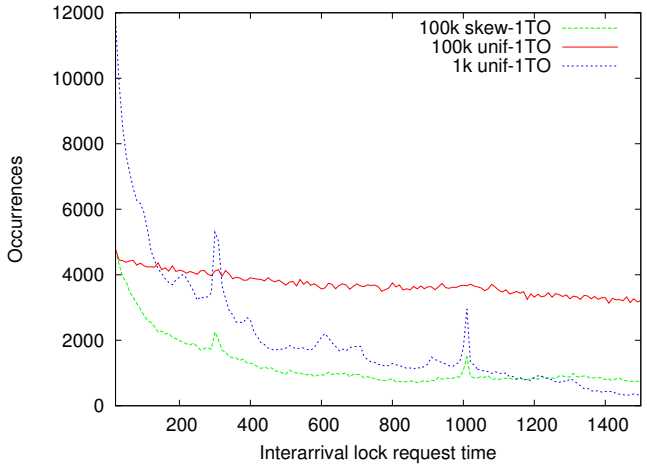


Fig. 3. Distribution of lock inter-arrival time using three different Radargun workloads.

*2) Low Level Statistics:* The set of additional low level statistics gathered from each individual Infinispan node (see [15] for an exhaustive list), is aimed to provide a detailed characterization of the performance and costs of the main subsystems involved in the processing of transactions along its life-cycle. These include both statistics (mean, and percentiles) on metrics typically used in SLAs (for instance, transaction execution time) and statistics useful for modelling purposes, such as the latency experienced by transactions along their various execution stages, the frequency of different types (write vs read) of transactions and of various contention-related events (e.g. successful vs failed lock acquisition). Among these, two types of statistics are particularly noteworthy:

**Probability distribution of lock inter-arrival time**: this information, encoded as an histogram, allows verifying whether one critical assumption holds for the applicability of Equation 1, namely, whether the lock arrival rate can be approximated by an exponential distribution. Equation 1, in fact, is guaranteed to hold only in case the lock requests arrival rate is poissonian, condition sufficient to ensure that the PASTA (Poissonian Arrival See Time Averages) property [16].

The data reported in Figure 3 shows an example of three lock inter-arrival time distributions that were obtained by configuring Radargun to generate transactions accessing data using different data access patterns (uniform vs skewed) on keysets of different sizes (1K vs 100K). As it can be seen the above parameters have a significant impact on the shape of the

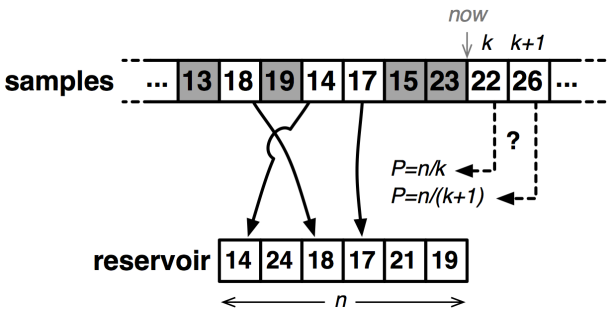


Fig. 4. Reservoir sampling algorithm [19] (Figure from [20]).

empirical lock inter-arrival time distributions, which present, at high skew or contention levels, spikes that are symptomatic of non-poissonian behaviors that can have an impact on the accuracy of the modelling methodology at the basis of the computation of the ACF.

By comparing, via Good of Fitness tests [17], the empirical lock arrival rate with (best-fitting) exponential distributions (or with other distributions for which the PASTA property holds, such as uniform distributions), one can therefore obtain a measure of the expected accuracy of the ACF in predicting the maximum degree of concurrency for a transactional application.

**Percentiles of transaction execution times**: percentiles are often preferred to simple averages in SLA negotiations as they provide more meaningful guarantees on the actual QoS delivered to the population of end users of a system. On the other hand, computing exact percentiles requires storing all the samples across the considered time window, or solving the problem of determining (statically or dynamically) an appropriate binning size [18].

In order to avoid the above complexity, we compute percentiles using Vitters reservoir sampling algorithm [21], which over time gives us an appropriate model for the distribution of the transaction execution lengths. Vitters algorithm fills an initially empty reservoir (array) of size $n$ with the first $n$ samples. Then, each $k$-th element is inserted in a random spot of the reservoir with a probability of $n/k$. This ensures an uniform sampling over the stream of data. The requested percentile is obtained by sorting the reservoir and picking the percentile of interest. For instance, to obtain the 95% of the transaction execution time we can simply read the value stored at index $j = n * 0.95$ of the sorted array.

*3) Thread Level Statistics:* The native statistics collection mechanism of Infinispan relies on a set of counters maintained by each node of the data grid. These counters are implemented by means of shared atomic variables that are updated (possibly concurrently) by threads upon the occurrence of relevant events. This approach to gather statistics has two main drawbacks:

- In many transactional applications, different threads have specialized transactional profiles (e.g. read vs write dominated workloads). By aggregated statistics at the data grid node level, it is impossible to capture statistical information that would allow for performing a detailed workload profiling on basis of activity of the different threads.
- On multi-core machines, atomic variables can increase the cache coherency traffic and impose the use of low-level atomic constructs (e.g. Compare and Swap), which, typically, rely on costly hardware operations, requiring, e.g., the generation of cache invalidation traffic or locking of system buses. The impact on system performance due to these factors may became relevant with some workload profiles and/or with high concurrency level, and may limit the system scalability.

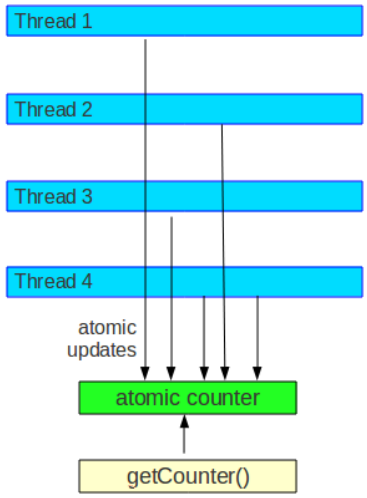


Fig. 5. Schema of the centralized statistics collection mechanism natively implemented in Infinispan.

On basis of the above motivations, the statistical data collection mechanism used in Infinispan has been extended, introducing, as a configurable alternative to the native centralized scheme, also a per-thread data gathering scheme. In the novel mechanism, each thread maintains a set of private copies of counters, one copy for each monitored metric. Upon the occurrence of an event that requires the update of a counter, the thread updates its own copy of the counter avoiding any kind of synchronization. This allows, on one hand, to gather differentiated statistics for each thread. Of course, when statistics at node level are needed, they can be still computed by collecting the values of the counters of the locally executing threads for the desired metric and by calculating the aggregated value (e.g. the average, the maximum, the minimum). In this implementation, the computation of an aggregated metric is performed when the metric is queried via its JMX interface. Figure 5 and Figure 6 provide, respectively, a comparison between the architectures of the collection mechanism used in Infinispan and the novel mechanism.

We conclude this section by presenting in Figure 7 the results of an experimental study aimed to assess the impact on Infinispan's performance due to the introduction of the new

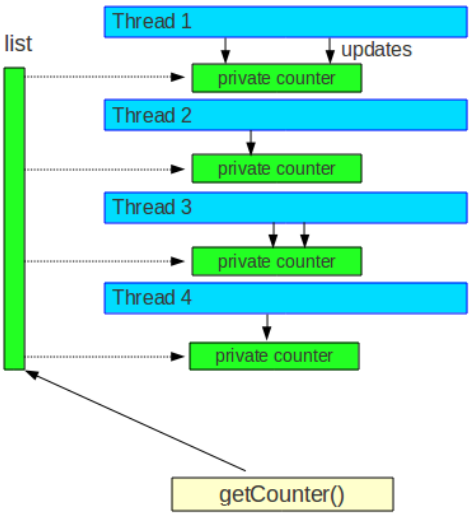


Fig. 6. Schema of the new per-thread statistics collection mechanism implemented in Infinispan.

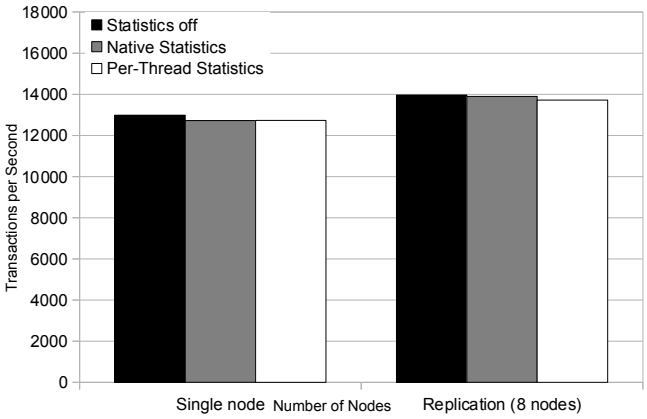


Fig. 7. Evaluating the overhead of the statistics collection mechanisms

set of statistics. To this end we injected a workload generating transactions with a very reduced conflict probability, and measured the throughput (committed transactions per second) achieved when running Infinispan in a single node and on 8 nodes (replication mode). The plots show that the throughput achieved by Infinispan when gathering the whole new set of statistics (implemented using the per thread collection scheme) is around 2% lower than when totally disabling the statistics collection system. This confirms the efficiency and feasibility of the proposed workload monitoring and analysis methodology.

### C. Workload and resource demand prediction

As already mentioned, the WA relies on the powerful $R$ statistical engine in order to perform various time series analysis. This is made possible by exploiting the recently introduced REST APIs of RHQ, which allows exporting the statistical data gathered from the monitored platform as time-series encoded in JSON [22] format.

Figure 8 shows a plot obtained computing 5% and 95% quantile, as well as 20-items simple moving average, of example workload data. The metrics are plotted in black, the average in blue, the 5% and 95% quantile in orange and green and with the help of the TTR library, the 50 samples moving average is plotted in red.

As a final remark, note that by exposing data via REST interfaces, the data gathered by RHQ can be straightforwardly provided as input to a plethora of machine learning tools, and not only to $R$. In fact, work is currently ongoing aimed to automatize data extraction from several popular machine learning tools, such as Cubist© [23] and Weka [24].

### D. QoS monitoring and alert notification

Cloud-TM's WA leverages on RHQ's advanced QoS monitoring and alert notification engine. The latter engine is designed to provide proactive notifications about events happening throughout the monitored platform. These events can be resources becoming unavailable, specific values for metrics being collected, resource configuration being changed, operations being executed, or even specific conditions found by parsing log events. As information flows into the RHQ system, it passes through the alerts processing engine. Here, the data can be transformed, filtered, or even correlated with data from other parts of the system. Users have full control over what they want to be notified about, and RHQ keeps a full audit trail of any conditions that have triggered alerts to fire. The alerts subsystem provides a wealth of different options for being notified proactively about potential issues in the system. As a result, it supports a breadth of different configuration options that allow for deriving very specific and customized semantics.

## IV. Conclusions

In this paper we presented an integrated system, tailored for transactional applications deployed in Cloud environments, allowing a detailed workload characterization and its accurate analysis. In particular we presented the Workload Analyzer, a key component of the Cloud-TM infrastructure that is in charge of automating workload characterization and providing estimates of future workload profile and resource demands. Further, the WA relies on the advanced alert processing engine of RHQ to generate alert signals upon the occurrence of QoS violations. The services provided by the WA are essential in order to extract from the Cloud-TM data grid information on whose basis the self-tuning policies driving the auto-configuration process of the platform can be defined.

We discussed the key design choices underlying the development of this component, which was built by combining, on one side, leading open source projects in the areas of systems management, statistical computing and stream-based algorithms, and on the other, innovative methodologies for what concerns the characterization of transactional applications.

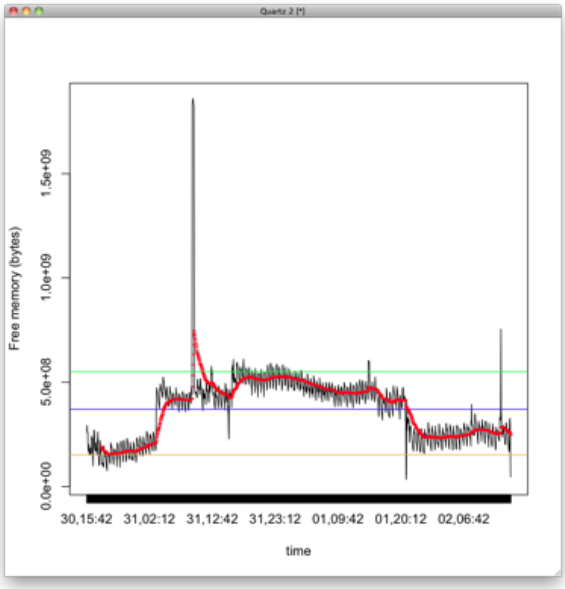


Fig. 8. Example plot of time series analysis obtained on data extracted via REST interfaces from RHQ.

## REFERENCES


[1] Cloud-TM, "Cloud-tm, a novel programming paradigm for the cloud." http://www.cloudtm.eu/.

[2] RHQ project - Red Hat, "The rhq project." http://www.rhq-project.org/.

[3] R-project, "The R-project." http://www.r-project.org.

[4] The stream-lib library, "A Java library for summarizing data in streams." https://github.com/clearspring/stream-lib.

[5] D. Didona, P. Romano, S. Peluso, and F. Quaglia, "Transactional auto scaler: Elastic scaling of nosql transactional data grids," Tech. Rep. 50, INESC-ID, December 2011.

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

[7] R. Palmieri, P. D. Sanzo, F. Quaglia, P. Romano, S. Peluso, and D. Didona, "Integrated monitoring of infrastructures and applications in cloud environments," June 2011.

[8] G. Box, G. Jenkins, and G. Reinsel, *Time series analysis: forecasting and control*. Wiley series in probability and statistics, John Wiley, 2008.

[9] A. Metwally, D. Agrawal, and A. E. Abbadi, "An integrated efficient solution for computing frequent and top-*k* elements in data streams," *ACM Trans. Database Syst.*, vol. 31, no. 3, pp. 1095–1133, 2006.

[10] Java / Oracle, "Java Management Extensions (JMX) Technology." http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html.

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

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

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

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

[15] Diego Didona, Pierangelo Di Sanzo, Roberto Palmieri, Sebastiano Peluso, Francesco Quaglia, Paolo Romano, Heiko W. Rupp, "Cloud-TM - Deliverable 3.2: Workload Analyzer." http://www.gsd.inesc-id.pt/~romanop/files/deliverables/D3_2.pdf.

[16] D. Konig, V. Schmidt, and E. A. Van Doorn, "On the pasta property and a further relationship between customer and time averages in stationary queueing systems," *Communications in Statistics. Stochastic Models*, vol. 5, no. 2, pp. 261–272, 1989.

[17] . W. D. Schunn, C. D., "Evaluating goodness-of-fit in comparison of models to data," *W. Tack (Ed.), Psychologie der Kognition: Reden and Vortrge anlsslich der Emeritierung von Werner Tack*.

[18] H. Shimazaki and S. Shinomoto, "A method for selecting the bin size of a time histogram," *Neural Computation*, vol. 19, no. 6, pp. 1503–1527, 2007.

[19] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, pp. 37–57, March 1985.

[20] W. Maldonado, P. Marlier, P. Felber, J. L. Lawall, G. Muller, and E. Riviere, "Deadline-aware scheduling for software transactional memory," in *DSN*, pp. 257–268, 2011.

[21] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

[22] D. Crockford, "Request for Comments 4627: The application/json Media Type for JavaScript Object Notation (JSON)." http://www.ietf.org/rfc/rfc4627.txt?number=4627.

[23] J. R. Quinlan, "Cubist." http://www.rulequest.com/cubist-info.html.

[24] E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, and I. H. Witten, *Weka: A machine learning workbench for data mining.*, pp. 1305–1314. Berlin: Springer, 2005.