Cloud-TM

Specific Targeted Research Project (STReP)

Contract no. 257784

Companion document for deliverable D3.4: Prototype of the Autonomic Manager

Date of preparation: 10 Jan 2013
Start date of project: 1 June 2010
Duration: 36 Months

Contributors

Bruno Ciciani,CINI
Joao Cachopo, INESC-ID
Maria Couceiro, INESC-ID
Diego Didona, INESC-ID
Pierangelo Disanzo, CINI
Stoyan Garbatov, INESC-ID
Roberto Palmieri, CINI
Sebastiano Peluso, CINI
Francesco Quaglia, CINI
Paolo Romano, INESC-ID
Diego Rughetti, CINI

Table of Contents

1 Introduction

This document is a companion of the software releases associated with deliverable D3.4 of the Cloud-TM project, namely the prototype implementation of the Autonomic Manager. The document presents design/development activities carried out by the project's partners in the context of WP3 (Task 3.3). The Autonomic Manager prototype is formed by three main subsystems, called Workload and Performance Monitor (WPM), Workload Analyzer (WA) and Adaptation Manager (AM). Further, the Autonomic Manager is in charge of supporting Quality-of-Service specification, and associated costs, which enable the definition and negotiation of determined QoS levels to be matched runtime by the Cloud-TM platform. The diagram in Figure 1 illustrates the main components forming the Autonomic Manager, highlighted via a gray box, and their collocation within the whole Cloud-TM architecture.

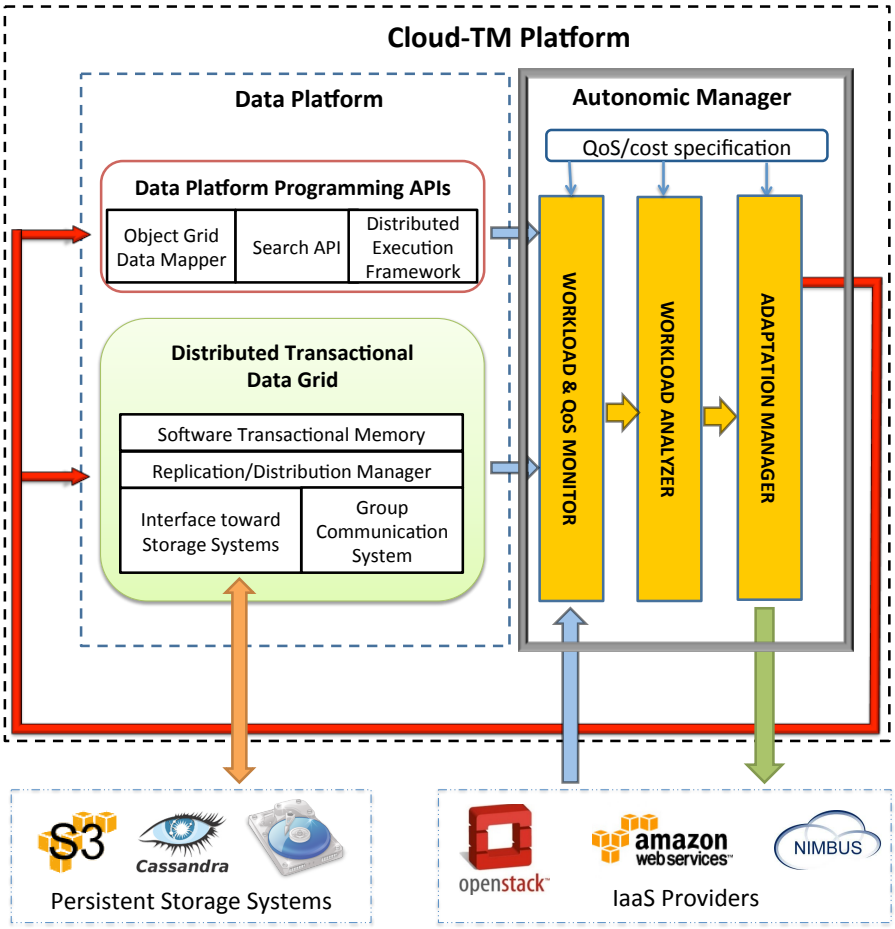

Figure 1: Architecture of the Cloud-TM platform.

The WPM and WA modules have already been the object of previous deliverables, namely deliverables D3.1 and D3.2. Hence this companion document is specifically focused on the AM module and on the supports for QoS/cost specification.

The structure of this document is the following. We start, in Section 2, by illustrating the set of features currently supported by the AM, which we classify into two main groups:

- tools for forecasting the performance of applications deployed on the Cloud-TM Data Platform, and aimed at identifying the optimal settings of several key configuration parameters (such as, platform's scale, data replication degree and replication strategy) taking into account pre-determined QoS/cost constraints;
- mechanisms aimed at maximizing the data access locality of applications deployed on the Cloud-TM Data Platform, by optimizing the dispatching of requests and the placement of data replicas.

Section 3 is devoted to presenting the performance forecasting tools currently integrated in the AM. The self-tuning mechanisms aimed at enhancing data locality are described in Section 4. Finally, in Section 6, we provide information on how to set-up the packages and present 6 demos aimed at illustrating usage examples of the self-tuning capabilities of the AM.

We note that, at the time of writing, the Cloud-TM Autonomic Manager is still being enhanced and extended to achieve smooth integration with the prototype of the Cloud-TM Data Platform (D2.3). Also, great effort has been spent in the last phase of the project on integrating, profiling and benchmarking the various modules forming the Autonomic Manager, most of which had to be developed independently to parallelize work in an effective way. Hence the current deliverable represents a snapshot of the current state of progress, rather than a final version of the Cloud-TM Autonomic Manager prototype, whose delivery is instead planned for end of May 2013 (D4.5 - Final Cloud-TM Prototype).

1.1 Relationship with other deliverables

The prototype implementation of the WPM has been based on the user requirements gathered in the deliverable D1.1 "User Requirements Report", and taking into account the technologies identified in the deliverable D1.2 "Enabling Technologies Report". The present deliverable has also relations with the following project deliverables:

- Deliverable D2.1 "Architecture Draft", where the complete draft of the architecture of the Cloud-TM platform is presented;
- Deliverable D3.1 "Prototype of the Workload Monitor" and deliverable D3.2 "Prototype of the Workload Analyzer". Particularly, these deliverables define the type of information, and the associated format, which can be exploited in input by the software components forming the AM;

- Deliverable D3.3 "Performance Forecasting Models", which provides the theoretical performance prediction background for the construction of actual AM components;
- Deliverable D2.3 "Prototype of the RDSTM and of the RSS" which provides the actual tunable protocols at the level of the Cloud-TM data platform.

2 Self-tuning features supported by the Cloud-TM Adaptation Manager

The diagram of Figure 2 provides a high level overview of the internal architecture of the AM, and reports the set of self-tuning mechanisms that it supports.

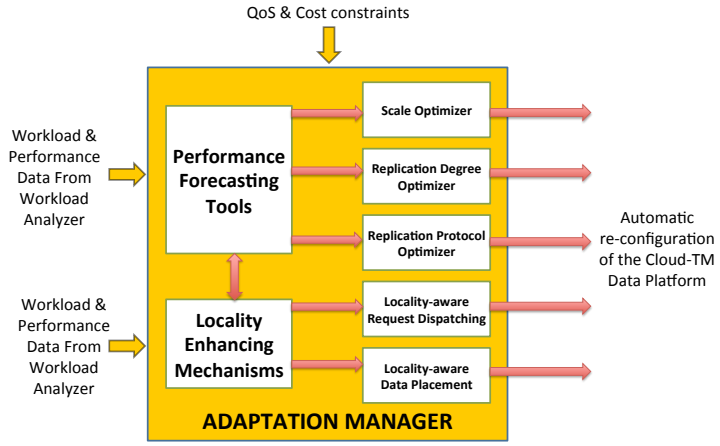

Figure 2: AM building blocks vs target estimated parameters.

As illustrated in Figure 2, the self-optimization mechanisms supported by the Cloud-TM Autonomic Manager 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;
   - number of replicas of each datum stored in the platform, which we also call, replication degree;
   - replication protocol (among the ones available within the data platform - see Deliverable D2.3);

   The core of the self-optimization processes responsible for tuning the above parameters/mechanisms consists of a set of performance forecasting models/tools, of which a detailed description was provided in Deliverable D3.3. These performance models are exploited by the AM in order to speculate on 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.

2. Mechanisms aimed at optimizing the data access locality of Cloud-TM applications, that is targeted at maximizing the collocation between the application logic

and the data it accesses. To this end two main mechanisms have been integrated in the Cloud-TM's AM:

- a mechanism 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 nodes and on the minimum/maximum number of copies maintained in the system;
- a mechanism 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.

It should be noted that the two classes of mechanisms discussed above can be employed either individually, or in synergy, depending on the degree of automation that a developer/administrator of an application deployed on the Cloud-TM platform aims at achieving.

3 Self-tuning techniques based on performance forecasting tools

Self-tuning modules integrated within the AM have been based on a suite of performance prediction results that have been achieved within the Cloud-TM project, most of which have been presented in deliverable D3.3. These have also been the object of the publications presented in [5–7]. These results rely on a plethora of differentiated prediction methodologies, which include:

- analytical methods;
- machine learning techniques;
- simulation techniques.

Given that these methodologies are highly complementary, their joint usage within Cloud-TM gave rise to the construction of an AM component entailing a wide spectrum of prediction (and hence optimization) capabilities. As for the employment of machine learning, these techniques provide AM with highly reliable prediction capabilities when considering system configurations (e.g. in terms of number of nodes within the underlying virtualized infrastructure) and workload profile/intensity falling within an already explored domain of the parameters' space. The exploration (and hence the associated training phase for the machine learner) could have been performed either off-line, or even on-line. Further, the training outcome can be refined along time.

On the other hand, machine learning techniques are known to provide limited extrapolation capabilities, hence not allowing for reliable prediction outside already explored domains. Since the number of tunable parameters within the Cloud-TM platform is relatively large (ranging from replication protocol selection, to the size of the underlying virtualized infrastructure, and to the actual placement of data copies across the nodes), the above limitation had to be mandatorily tackled by the AM component. This has been done thanks to the reliance on analytical and simulative models.

As for the former ones, they are typically less computationally demanding than simulative approaches. On the downside, the design phase of analytical models is normally significantly more onerous. Also, due to their inherently higher complexity, analytical approaches tend to introduce a higher number of simplifying assumptions, which may ultimately degrade their prediction accuracy. Regarding the simulation performance, the simulation framework we have designed and implemented as one of the components of AM has been explicitly structured such in a way to be compliant with the high performance parallel simulation engine made available via the ROOT-Sim open source project [14]. Hence, AM also entails high performance simulation capabilities, which make very large models tractable and solvable within significantly reduced computation time.

As a final note, the released AM implementation also entails components where the above techniques/methods are exploited in a synergical (rather than orthogonal) manner. Particularly, AM offers a hybrid prediction module based on both analytical and machine learning techniques. Also, in our tool design/development, there is no impediment to use the outcome of machine learning in relation to specific parameters

as the input to simulation models. Particularly, when setting up simulation models, a modeling approach for network latencies is required. As for this aspect, the wide literature on network models and the wide set of tools available for network simulation target traditional networks, not those that interconnect virtual machines (maybe located onto the same physical node). As a consequence, the reuse of traditional network models may give rise to some prediction discrepancy. To overcome this drawback, the network model to be used as a component of the whole distributed data platform simulation model can be derived via delay estimation performed via machine learning schemes.

Details on all the software packages included within AM, which are aimed at self-optimization via the reliance on performance forecasting models/schemes, are presented in the remainder of this section.

3.1 Artificial neural network predictor

The Artificial Neural Network (ANN) prediction module included within AM is a black-box prediction subsystem, which is able to estimate performance indexes for a Cloud-TM application transparently to the actually selected replication protocol. The input features taken into consideration for ANN are: (i) the number of nodes within the platform, (ii) the degree of replication of data and (iii) the (average) number of concurrently active clients (expressing the workload intensity). Hence, with respect to the scheme shown in Figure 2, ANN constitutes an implementation of parts of both the Elastic Scaling Manager and the Data Platform Optimizer since it targets performance prediction vs both the scale of the platform and the degree of data replication. The number of active clients is not under the direct control of ANN, though. In fact, the value of this parameter is a reflection of application proper dynamics.

A schematization of the internal structure of ANN and of the relations between the ANN module and other Cloud-TM components is shown in Figure 3. Particularly, the above three input features for the estimation process are acquired by ANN by directly interacting with WA. This interaction is based on a proper adapter that is in charge of parsing the data provided by WA and of translating their representation to a convenient form, matching the expectation of the core software modules implementing ANN. On the other hand, the output provided by ANN is the actual estimation of the target performance parameters (within the learning domain), namely the throughput and the (transaction) execution latency.

The whole package has been developed using the `C` language as the core technology. Also, the actual implementation of the ANN has been based on fully connected networks, with three layers each. In our implementation, the training algorithm has been implemented within a module external to the core module implementing the neural network, and has been based on back-propagation [3, 21, 22]. This provides a level of modularity that allows changes in the training rules, while imposing no change to the actual network implementation. Anyway, for the actual implementation, as also discussed in [6], we observed that a number of hidden layers equal to one was a good trade-off between prediction accuracy and learning time. In this case, the number of hidden nodes for which the networks provided the best approximations was on the order of 16.

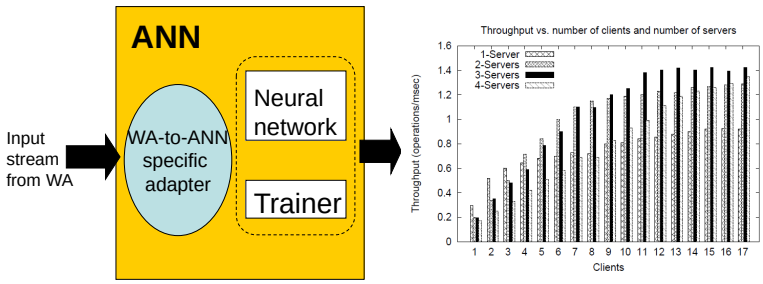

Figure 3: The ANN package.

We highlight that the ANN module's operation is completely transparent to the currently selected replication protocol, among those offered by the Cloud-TM data platform. This leads to the scenario where the prediction of the performance while varying the replication protocol can be carried out in two alternative ways. On one hand, learning and estimation of the performance parameters could be explicitly performed multiple times via ANN, by activating a different replication protocol in each phase. On the other hand, the estimation performed by ANN when considering, e.g., a default replication protocol, can be successively refined via the complementary techniques offered by AM, which we shall discuss in the subsequent sections.

Pointers to source code and additional documentation:

- The source code of ANN is available at the following URL:

  ```
  https://github.com/cloudtm/cloudtm-autonomic-manager/
  tree/master/src/ann
  ```

  as well as in the deliverable's virtual machine image at the path :

  ```
  ~cloudtm/cloudtm-autonomic-manager/src/ann/
  ```

  Detailed technical information on ANN can be found in the following technical report [6].

3.2 MorphR predictor

The MorphR prediction module included within AM follows a complementary approach, with respect to ANN. In fact, it implements a machine learning based predictor where, for any given number of nodes and replication degree, the actual protocol used for data replication and replica synchronization is explicitly taken into account. The approach taken by MorphR is based on classification, and the implementation of the

classification scheme is based on the state of the art C5.0 classifier [19]. At training time, C5.0 builds a decision-tree classification model via a greedy heuristic used to partition, at each level of the tree, the training data-set according to the input feature that maximizes information gain. The output model is a decision-tree that closely classifies the training cases according to a rule-set, which can then be used to classify (future) scenarios, namely to decide the best performing replication strategy.

Complementarity with respect to ANN is also reflected in the amount of input features selected for the classification process. In particular, in MorphR, a set of 14 features has been selected to provide a detailed characterization of:

- the transactional workload: percentage of write transactions, number of read and write operations per read-only and write transactions and their local and total execution time, abort rate, throughput, lock waiting time and hold time.
- the average utilization of the computational resources of each node and of the network: CPU, memory utilization and commit duration.

Again in relation to the integration with WPM and WA, similarly to ANN, the MorphR module is based on an adapter that translates the representation of samples coming from WA to the one suited for the internal organization of MorphR software. The latter has been realized by relying on Java technology.

MorphR can be used as a stand-alone tool within AM. On the other hand, it could also be used in a synergic manner with respect to ANN. Particularly, once selected a default replication protocol for the training phase, ANN can be used to determine reference well-suited intervals of values for the number of servers and the degree of data replication in relation to the target SLA. Then MorphR can refine the estimation process within the domain related to the selected intervals by investigating on the effects of replication-protocol selection. Also, by explicitly considering features associated with the transactional workload (e.g. the percentage of write transactions), MorphR can improve performance prediction accuracy in all the scenarios where the workload exhibits phase-behaviors.

Pointers to source code and additional documentation:

- The source code of MorphR is available at the following URL:

  ```
  https://github.com/cloudtm/cloudtm-autonomic-manager/
  tree/master/src/morphr/
  ```

  as well as in the deliverable's virtual machine image at the path :

  ```
  ~cloudtm/cloudtm-autonomic-manager/src/morphr/
  ```

  Detailed technical information on MorphR can be found in the following technical report [4].

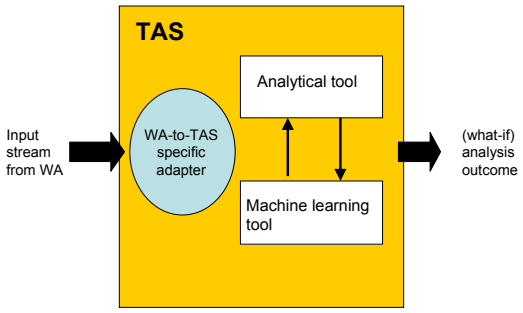

Figure 4: The TAS package.

3.3 Transactional auto-scaler predictor

The Transactional Auto-Scaler module (TAS) represents the implementation of a hybrid prediction methodology where analytical and machine learning approaches are used in combination. The analytical part of the tool is focused on predicting the effects on performance due to data contention. On the other hand, the machine learning part is oriented to capture the effects associated with contention of infrastructure resources, such as CPU and network.

More in detail, the analytical model supported by TAS uses mean-value analysis techniques to forecast the probability of transaction commit, the mean transaction response time, and the maximum system throughput. This allows supporting what-if analysis on parameters like the degree of parallelism (number of nodes) in the system or shifts of workload characteristics, such as changes of the transactions' data access patterns. One key element of the modeling approach is that it does not rely on any a priori knowledge about the probability of a write operation to insist on a specific datum, thus not requiring specific assumptions on the distribution of the accesses (such as uniformity of the accesses over the whole data-set). Instead, it introduces a powerful abstraction that allows the on-line characterization of the application data access pattern in a lightweight and pragmatical manner, which is called Application Contention Factor (ACF), whose analytical details have been presented in [7].

On the other hand, the machine learner integrated within TAS is based on Cubist [20], which is a decision tree regressor that approximates non-linear multivariate functions by means of piece-wise linear approximations. As hinted, machine learning has been used by TAS in order to estimate the effects on performance due to contention on infrastructure resources. In particular, this has been done by providing estimations on two key performance indexes, namely the time required to perform any local operation (such as data read/write) on any node (which also reflects CPU contention), and the time required for the execution of distributed coordination actions across the nodes within the platform, such as when running the 2PC protocol (which also reflects network level contention).

The architecture of TAS is schematized in Figure 4, where a feedback interaction is clearly shown between the two analytical and machine learning tools. The analytical tool queries the machine learning tool in order to get estimations of some performance parameters that are directly affected by contention on infrastructure resources. These estimations are used by the the analytical tool as input values to forecast the main system performance parameters. On the other hand, it can capture shifts in the data contention pattern, which may give rise to a different usage of infrastructure resources. This information can be provided in input to the machine learning tool. As a last note, given the presence of the analytical component, TAS is able to perform predictions outside the machine learner training domain (as an example, a prediction can be done on what would be the expected performance when running on top of a platform whose size was not considered in the training phase). On the other hand, the input values required from the machine learner in such scenarios can be determined via extrapolation capabilities provided by the machine learner.

Pointers to source code and additional documentation:

- The source code of TAS is available at the following URL:

  ```
  https://github.com/cloudtm/cloudtm-autonomic-manager/
  tree/master/src/tas
  ```

  as well as in the deliverable's virtual machine image at the path :

  ```
  ~cloudtm/cloudtm-autonomic-manager/src/tas/
  ```

  Detailed technical information on TAS can be found in the following paper [7].

3.4 Simulation framework

The simulation framework for transactional data grid platforms that has been developed within the context of the Cloud-TM project, which we refer to DAGS (DAta Grid Simulator), has the ability to predict performance indexes while varying the following set of parameters:

- number of nodes within the platform;
- degree of data replication;
- placement of data copies.

The framework can operate via the employment of models of the workload (e.g. in terms of distribution of the data accesses onto the data-set), which are already implemented within the offered simulation modules, or according to a trace based approach. Particularly, a DAGS adapter module has been implemented in order to gather data from WA and to determine the actual input parameters for the simulation runs.

The framework has been developed as a library implementing data grid models developed according to the traditional event-driven approach, where the evolution of

each individual entity to be simulated within the model is expressed by a specific event-handler. Also, give that model execution performance is one of the targets for DAGS, the core technology used for building the framework is the `C` programming language. Similarly to other tools included within AM, the framework is oriented to flexibility, and to the possibility of enlarging, in an easy way, the set of modeled coordination/synchronization protocols at the level of the data platform. This has been achieved by designing the framework via a skeleton model that can be easily extended/specialized to deal with different data management logics. This would allow to easily cope with simulating additional data management mechanisms beyond those already by the Cloud-TM data platform.

At the bottom-line, model execution can be carried out in a sequential fashion, by relying on a calendar queue event-scheduler offered within the package. On the other hand, again with simulation performance as target, the implemented ANSI-C event-handlers have been designed in order to be compliant with the programming API supported by the ROOT-Sim open source project, which offers a free environment (engine) for high performance model execution on top of both parallel and distributed platforms. Partitioning of the simulation model on the available computing resources is handled by the ROOT-Sim environment in a transparent way. This opens a spectrum of possibilities in relation to the trade-off between simulation time (hence the time required for the prediction of performance indexes by the Autonomic Manager) and the amount of resources dedicated to the simulation process. Particularly, when the scale of the simulation model gets largely increased, AM can in its turn acquire additional computational resources from the cloud (to be exploited via the ROOT-Sim platform) in order to speedup the prediction phase, thus favoring, e.g., the timely delivery of the results of the QoS negotiation phase to the customer.

Further, the DAGS framework can be used to perform what-if analysis in a complementary manner with the respect to the TAS module, especially when predictions need to be carried out outside of the domain where the machine learning module within TAS has been trained. This might be the case when supporting what-if analysis for largely increased system scales, for which highly reliable training would require the employment of a large amount of resources to be acquired by the Autonomic Manager.

Pointers to source code and additional documentation:

- The source code of DAGS is available at the following URL:

  ```
  https://github.com/cloudtm/cloudtm-autonomic-manager/
  tree/master/src/dags/
  ```

  as well as in the deliverable's virtual machine image at the path :

  ```
  ~cloudtm/cloudtm-autonomic-manager/src/dags/
  ```

  Detailed technical information on DAGS can be found in the following technical report [5].

4 Locality enhancing mechanisms

The Cloud-TM Autonomic Manager maximizes efficiency, not only by adjusting the scale and replication options of the Cloud-TM Data Platform, but also via mechanisms aimed at enhancing the access locality of user applications by means of two, complementary self-tuning mechanisms:

- The first mechanism, called AUTOPLACER, optimizes the placement of replicas of data across the nodes of the Cloud-TM Data Platform, in order to minimize the frequency with which applications access remote data. The mechanism operates in rounds, optimizing the placement of user-tunable number of data items per round, in order to minimize the overheads associated with data relocation. The mapping of data replicas to nodes is adjusted based on the data access patterns generated by the nodes of the platform, while keeping into account constraints on the capacity of nodes and on the number of copies maintained in the system.

- While AUTOPLACER exploits the locality in the access patterns generated by applications (by accordingly optimizing the placement of data in the platform), the second mechanism, which we call Locality-aware Transaction Dispatching (LTD), aims at increasing the locality of the data access streams generated by applications.

  To this end, LTD dispatches applications' requests based on the correlation of their working sets, i.e. the expected overlapping between the data sets that will be accessed while processing an application request. LTD partitions requests into disjoint groups, based on the contents of their working sets, and assigns to a node of the Cloud-TM data platform only a particular subset of request groups.

  By collocating on the same node the processing of requests types having highly correlated working sets, LTD allows enhancing the locality of the data accesses generated on each node of the Cloud-TM platform, paving the way for an effective optimization of the data placement via AUTOPLACER.

In the following we describe AUTOPLACER and LTD in more detail.

4.1 Self-tuning data placement

AUTOPLACER is a system aimed at self-tuning the placement of replicas of data items (i.e., key/value pairs) within the Cloud-TM data platform. We recall that Infinispan (namely the transactional data grid that represents the backbone of the Cloud-TM data platform), relies, by default, on simple, random hash functions to disseminate data across nodes. This approach allows lookups to be performed locally, and guarantees that the join/leave of a node incurs in a limited change in the mapping of keys to buckets. However, due to the random nature of data placement (oblivious to the access frequencies of the node to data), a significant amount of remote accesses may be experiences, which may slow down the system.

AUTOPLACER addresses this problem by identifying the data items having a sub-optimal placement in the platform and re-locating them automatically to maximize

access locality. To this end, AUTOPLACER employs a lightweight self-stabilizing distributed optimization algorithm, whose key phases are illustrated in the diagram in Figure 5. The algorithm operates in rounds, and, in each round, it optimizes the placement of the top-k "hotspots", i.e. the objects generating most remote operations, for each node of the system. Not only this design choice allows reducing the number of decision variables of the data placement problem (solved at each round) and ensuring its practical viability, it also abates the monitoring overhead for tracking and exchanging data access statistics.

In order to minimize the overhead for identifying the "hotspots" of each node, AUTOPLACER adopts a state of the art stream analysis Space-Saving Top-k algorithm [16] that permits to track the top-k most frequent items of a stream in an approximate, but very efficient manner. The information provided by the Space-Saving Top-k algorithm is then used to instantiate the data placement optimization problem, which is partitioned in independent subproblems and solved in parallel by the nodes of the platform.

In order to guarantee 1-hop routing latency, AUTOPLACER does not rely on dedicated directory services. Conversely, AUTOPLACER combines the usage of consistent hashing (which is used as the default placement strategy for less popular items) with a highly efficient, probabilistic mapping strategy that operates at the granularity of the single data item, achieving high flexibility in the relocation of (a possibly very large number of) hot data items.

The key solution introduced to pursue this goal is a novel data structure, named *Probabilistic Associative Array* (PAA), which is aimed to minimize the cost of encoding the association between the set of keys to be re-located and the corresponding target nodes. PAAs expose the same interface of conventional associative arrays, but, in order to achieve space efficiency, they use probabilistic techniques and trade-off accuracy, and can reply erroneously to queries with a user-tunable probability p. Internally, PAAs rely on Bloom Filters (BFs) and on Decision Tree (DT) classifiers. BFs are used to keep track of the elements inserted so far in the PAA; DTs are used to infer a compact set of rules establishing the associations between keys and values stored in the PAA. In order to maximize the effectiveness of the DT classifier, we define an intuitive programmatic API that allows programmers to provide additional information on the keys to be stored in the PAA (e.g., the data type of the object associated with the key). This information is then exploited, during the learning phase of the PAA's DT, to map keys into a multi-dimensional space that can be more effectively clustered by a DT classifier.

Once that the PAAs encoding the new mapping of data to nodes has been computed, they are disseminated among all nodes. The final task performed in each optimization round consists in physically relocating the data items for which new locations have been derived. This result is achieved by triggering the state transfer to move the objects according to the relocation map encoded in the PAAs.

The AUTOPLACER systems has been integrated with the other layers belonging to the Autonomic Manager, namely WPM and WA. This is done by including the top-k information within the stream of data provided by WPM/WA to the components belonging to AM. On the other hand, the production of top-k information entirely resides within the Cloud-TM data platform, which exports it towards the WPM via JMX.

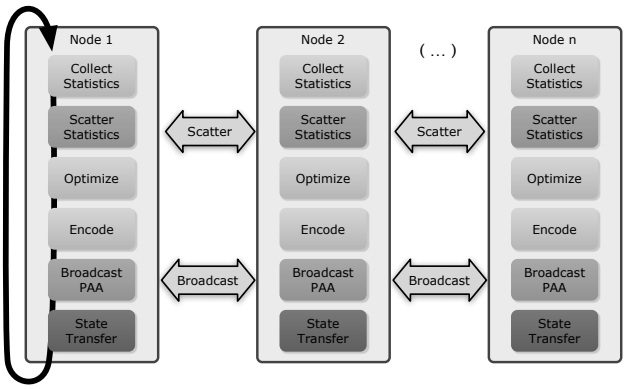

Figure 5: The AutoPlacer system.

Pointers to source code and additional documentation:

- The source code of the AUTOPLACER is available at the following URL:

  ```
  https://github.com/cloudtm/cloudtm-autonomic-manager/
  tree/master/src/autoplacer
  ```

  as well as in the deliverable's virtual machine image at the path :

  ```
  ~cloudtm/cloudtm-autonomic-manager/src/autoplacer/
  ```

  Detailed technical information on AUTOPLACER can be found in the following technical report [17].

4.2 Locality-aware Transaction Dispatching

As hinted in Section 4, the Cloud-TM Autonomic Manager allows also to self-tune the policy used to dispatch requests towards the nodes of the Cloud-TM Data Platform in order to enhance the locality of the data access patterns generated by applications.

The mechanism in charge of this self-tuning process is the, so called, Locality-aware Transaction Dispatching (LTD) algorithm. The key idea at the basis of the LTD algorithm is to dispatch requests that have a high probability of accessing common data on the same (set of) node(s). By processing on the same nodes requests having highly correlated working sets, the efficiency of the Cloud-TM Data Platform is increased in a number of ways.

In scenarios in which the Cloud-TM Data Platform is deployed using a partial replication strategy, the enhancement of the access locality at each node paves the way for the identification of highly optimized data placement layouts by AUTOPLACER. This translates into a reduction of the frequency of remote operations to fetch data maintained on different nodes, and in the reduction of the number of nodes to be contacted when committing a transaction. These two factors have a strong impact on the network traffic generated by applications deployed on the Cloud-TM platform, playing a fundamental role in determining the performance, scalability and operational costs [1].

In scenarios in which the Cloud-TM Data Platform is deployed using a full replication scheme, in which all data is maintained in memory, an enhancement of the locality has still a strong impact on the effectiveness of the caching mechanisms employed at the operating system and CPU level.

Finally, in both deployment modes (full vs partial replication), processing requests with highly correlated working sets in the same node has also a beneficial effect on the efficiency of the concurrency control mechanism used to regulate transaction processing in the Cloud-TM Transactional Data Grid. In fact, conflicts between concurrent transactions originated on the same node can be detected and handled much more efficiently than for the case of remote transactions: conflicts between local transactions are manageable using lightweight local validation schemes, whereas conflicts between remote transactions can only be detected using expensive distributed coordination schemes.

The LDT algorithm operates in two phases. In the first phase, it gathers information on the data access patterns generated by applications developed using the Cloud-TM Object Grid Data Mapper module (OGDM), i.e. the Fenix Framework (FF). This is done in a totally transparent way for the application developers, by injecting, during the compilation phase of the applications' DML, extra code into the target application. This code contains call-backs to the, so called, Data Access Pattern module of FF, which is responsible for updating the statistical information whenever an access (read-/write operation) is performed over a domain instance, as well as code for identifying the context within which the operations are taking place.

At run-time, when a representative volume of behaviour information has been collected, the DAP analyzes and predicts future data access patterns' behaviour by employing one of three alternative stochastic model implementations - Bayesian Updating [8, 9], Markov Chains [9, 10] and Importance Analysis [9, 11].

[1] In several IaaS platforms applications are charged based on the amount of network traffic they generate.

Once the models have been learnt, LTD relies on the Latent Dirichlet Allocation partitioning algorithm [13] to subdivide the various application's request types into disjoint groups, such that the correlation between the working sets of all request types placed in a particular group is maximized.

Pointers to source code and additional documentation:

- The source code of the LTD is available at the following URL:

  ```
  https://github.com/cloudtm/cloudtm-autonomic-manager/
  tree/master/src/ltd
  ```

  as well as in the deliverable's virtual machine image at the path :

  ```
  ~cloudtm/cloudtm-autonomic-manager/src/ltd/
  ```

  Detailed technical information on the LTD algorithm, and on the set of technologies it leverages can be found in the following papers [8–11].

5 QoS and cost management

The supports for QoS and cost management offered by the Autonomic Manager can be used (A) to allow the customer to provide information about the application logic to be hosted on top of the Cloud-TM platform, to indicate the associated expected workload and its features, and to indicate expected performance values (to be matched within the SLA), and (B) to inform the customer about the existence of an operating configuration of the Cloud-TM platform capable of satisfying that SLA, and at what cost. exists, which is able to satisfy that SLA, and at what cost.

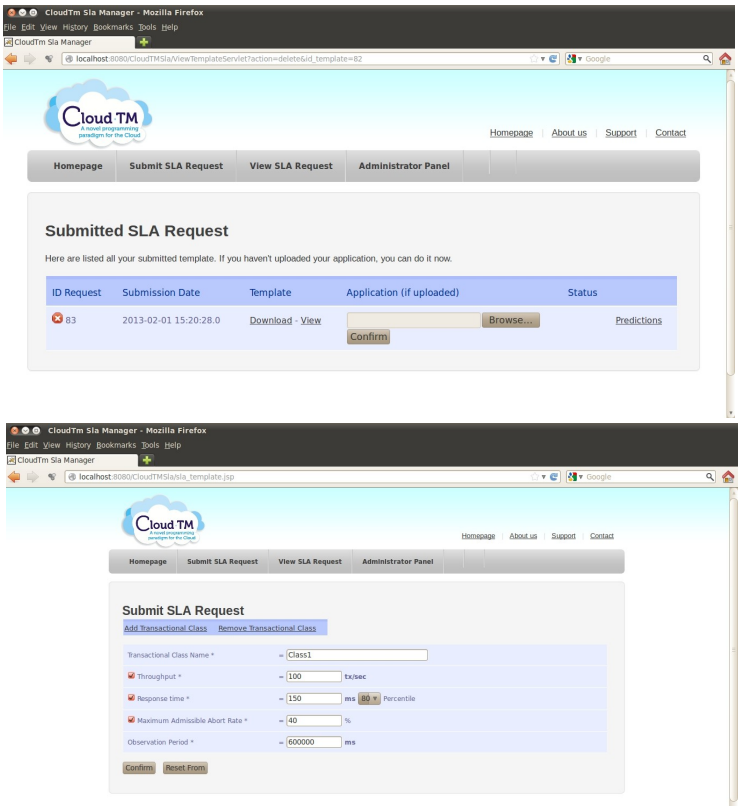

Figure 6: Example screenshots in relation to the web upload of the application code by the customer and the specification of requested performance levels.

The whole architecture of the component in charge of managing the QoS negotiation process relies on the exploitation of results provided by other EU projects operating in this same field, particularly the SLA[AT]SOI project. Essentially, the whole application relies on web technologies, thus providing the possibility to the customer to access the system via browser. Example screenshots in relation to the web interface provided to the customer are reported in Figure 6. At the back-end, the exploited technology is jsp/servlet (with the ajax framework), which has been used to built modules that are

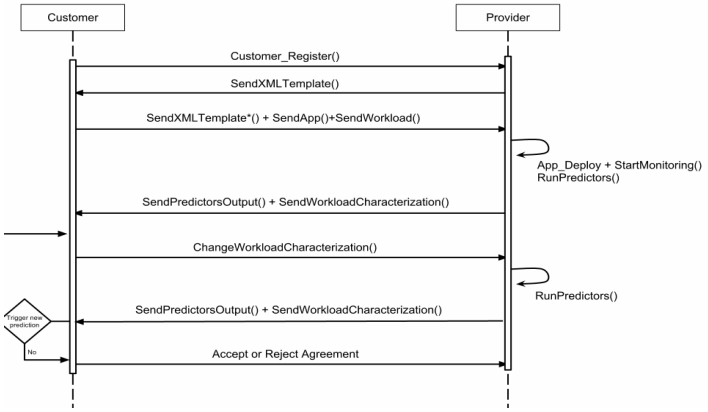

Figure 7: The QoS negotiation process.

able to perform access and manipulation to records maintained by a MySQL database.

The database records log information associated with a specific customer, which is expressed in terms of details in relation to the customer requests and the outcome of the QoS negotiation process. State transitions within this process can be actuated according to the iterative scheme shown in Figure 7, where the customer is allowed to reconsider/tailor it original request on the basis of the (tentative) negotiation outcome by the provider. Obligations and costs related to the QoS negotiation process are recorded by relying on an WML template, which is a particular XML structure defined within the context of the SLA[AT]SOI project. The generation of WML templates allows any delivered information to be manageable by any other QoS-oriented systems adhering to the specifications provided by SLA[AT]SOI.

Pointers to source code and additional documentation:

- The source code of the framework for QoS management is available at the following URL:

  ```
  https://github.com/cloudtm/cloudtm-autonomic-manager/
  tree/master/src/qos
  ```

  as well as in the deliverable's virtual machine image at the path :

  ```
  ~cloudtm/cloudtm-autonomic-manager/src/qos/
  ```

6 Prototype setup and usage examples

The packages forming the AM are distributed as a virtual machine (VM) image of type qcow2, ready to be deployed on KVM-based IaaS platforms, such as OpenStack. Of course the image can be readily converted to run on other hypervisors (e.g., XEN) using open-source tools like qemu-image[2].

The VM comes with a default user (the `cloudtm` user), with a password-less account. The contents of the software packages are stored in the home directory of the `cloudtm` user, and are organized according to the following directory structure:

```
cloudtm-autonomic-manager
    |-docs/
    |    |-ann/
    |    |    \-ann_reference.pdf
    |    |-autoplacer
    |    |    \-autoplacer.pdf
    |    |-morphr/
    |    |    \-morphr_reference.pdf
    |    |-tas/
    |    |    \-tas.pdf
    |    |-dags/
    |    |    \-dags_reference.pdf
    |    |-qos/
    |    |    \-qos_reference.pdf
    |-demos/
    |    |-Demo1-ANN/
    |    |-Demo2-MorphR/
    |    |-Demo3-TAS/
    |    |-Demo4-DAGS/
    |    |-Demo5-AutoPlacer/
    |    \-Demo6-LTD/
    \-src/
         |-ann/
         |-autoplacer/
         |-dags/
         |-ltd/
         |-morphr/
         |-qos/
         \-tas/
```

The *cloudtm-autonomic-manager* folder contains the source code, documentation and examples of the preliminary prototype of the Cloud-TM Autonomic Manager. Specifically:

- The *docs* folder contains reference documentation for each module of AM and this companion document.
- The *demos* folder contains the source code of the demo applications.
- The *src* folder contains the source code of each module belonging to the Cloud-TM Autonomic Manager.

[2] `http://www.qemu.org`

6.1 Installing and running the demo applications

In this section we describe how to install, configure and execute a set of applications that demonstrate the usage of the self-tuning capabilities of the Cloud-TM Autonomic Manager.

6.1.1 Demo 1 - Automated Elastic Scaling (based on ANN)

In this section we discuss how to install and run an example application demonstrating the usage and the actual run-time dynamics produced by the ANN component. For this example we adopt a distributed benchmark, named RadrGun Client Server, that executes TPC-C [23] transactional profiles against the Cloud-TM data platform in such a way that the TPC-C transactions can run as clients on separate virtual machines wrt the machines hosting the data platform. In particular the data platform is formed by a clustered set of Cloud-TM instances, each one of them running as a server on a single virtual machine and accepting transactional requests from the clients threads.

Every virtual machine executing a data platform instance is monitored by means of WPM/WA, which is responsible for gathering (via Consumers and Producers instances) and aggregating statistics (via a Log Service instance) produced by the data platform during the execution of the benchmark, and making available those statistics to all the components of the Autonomic Manager. In particular, for this purpose we adopt a push mechanism according to which the ANN component can register a subscription to WPM/WA on a set of virtual machines S, and it is notified whenever the WPM's Log Service component logs a sample of statistics produced by the Cloud-TM instances running on S.

When the ANN instance receives new statistics, it can decide to change the actual configuration of the Cloud-TM data platform, namely the number of data platform instances and the data replication degree, on the basis of the current workload, i.e. the average number of active transactional threads executing on the Cloud-TM data platform, with the purpose of maximizing the overall system throughput, i.e. the average number of committed transactions per second, or minimizing the transactional response time, i.e. the average time spent to successfully execute a transactional request. This change in configuration is executed by an actuator module we refer to as Autonomic Manager Actuator, which is responsible for (i) stopping subsets of instances belonging to the Cloud-TM data platform, (ii) starting new data platform instances, (iii) triggering a change of the data replication degree within the Cloud-TM cluster.

Configuring and running the example

The current example is already shipped in the VM image and it is ready to be executed, for less than a configuration process. However, for the sake of self-containment, we list in what follows the steps that should be performed in order to download, compile, configure and run the application.

The application requires:

- Java 6
- Maven 3.0.3

- Ant 1.8.4

The package can be downloaded at the following URL

```
https://github.com/cloudtm/cloudtm-autonomic-manager/
tree/master/demos/Demo1-ANN
```

and it has the following structure

```
Demo1-ANN
 |-ann/
 |-AutonomicManagerActuator/
 |-DataGridClient/
 |-DataGridServer/
 |-RadargunClientServer/
```

As a prerequisite to run the example application we need the WPM runtime and a WPM Connector (adapter) to be used by the ANN component for registering subscriptions on and acquiring new statistics from WPM/WA. The Connector's source code is available at the following URL

```
https://github.com/cloudtm/Workload_Monitor_Connector
```

and it can be compiled using the following commands:

```
$ cd Workload_Monitor_Connector
$ ant
```

The WPM's source code is available at the following URL

```
https://github.com/cloudtm/wpm
```

and it can be compiled using the following commands:

```
$ cd wpm
$ ant
```

In order to compile ANN, the AutonomicManagerActuator and the DataGridServer modules, we have to enter their main folders and to execute the ant command:

```
$ cd ANN
$ ant
$ cd ../AutonomicManagerActuator
$ ant
$ cd ../DataGridServer
$ ant
```

Since the ANN module needs to connect to the actuator as well as to WPM, we have to move the output of the AutonomicManagerActuator compilation, namely the AutonomicManagerActuator/AutonomicManagerActuator.jar, and the output of the Workload_Monitor_Connector compilation, namely Workload_Monitor_Connector/WPMConnector.jar to the ANN/lib folder.

Given that the adopted benchmark runs the transactional threads as clients of the data platform servers, the RadargunClientServer depends on a client stub used to transparently implement the communication protocol between a benchmark thread and a server thread of the data platform. For this reason, right before compiling the RadargunClientServer, we need to compile and install in the local maven repository the DataGridClient stub according to the following steps:

```
$ cd DataGridClient
$ ant
$ mvn install:install-file -Dfile=InfinispanClient.jar
-DgroupId=simutools.infinispan.client -DartifactId=infinispan-client
-Dversion=1.0.0 -Dpackaging=jar
```

Then RadargunClientServer can be compiled via maven by executing the following commands:

```
$ cd RadargunClientServer
$ mvn install
```

A run of the example application is divided into two phases. During the first phase we run and populate an instance of the Cloud-TM data platform by creating the initial TPC-C data-set. During the second phase we (i) start-up the remaining instances of the data platform that can join the same cluster and share the same initial data-set, and (ii) execute the TPC-C benchmark on the platform.

From now on we suppose to have two types of virtual machines with two different roles. On a virtual machine of type $master$ we will run the WPM's Log Service, the Autonomic Manager Actuator and the ANN module, while on a virtual machine of type $slave$ we will run a benchmark client, or a data platform instance monitored by WPM. Furthermore, in this example, we consider to run one machine of type $master$ having $IPmaster$ as IP address; on the other hand we consider to run two sets of machines of type $slave$: the first set $S1$ having IP addresses $IPslave1$, $IPslave2$, $IPslave3$, $IPslave4$, which will host the data platform, and the second set $S2$ having IP addresses $IPslave5$, $IPslave6$, $IPslave7$, $IPslave8$, which will host the benchmark clients.

To configure the connection between the benchmark clients and the data platform instances we need the following line in the cacheprovider.properties configuration file (see Listing 1) of the RadargunClientServer folder.

Listing 1: cacheprovider.properties configuration file

```
remote IPslave1 IPslave2 IPslave3 IPslave4
```

A. Data platform population

To run the population phase of the benchmark, we have to execute a population client thread on an instance of the data platform. To this end, we execute the following commands in order to run a data platform instance on the machine $IPslave1$:

```
$ cd DataGridServer
$ ./bin/runServer.sh
```

Then we change the benchmark.xml configuration file contained in RadargunClientServer/target/distribution/RadarGun-1.1.0-SNAPSHOT as shown in Listing 2, and we execute the following commands on machine $IPmaster$:

```
$ cd RadargunClientServer/target/distribution/RadarGun-1.1.0-SNAPSHOT
$ ./bin/benchmark.sh IPslave5
```

Listing 2: Benchmark configuration for TPC-C population

```
<bench-config>
   <master bindAddress="${127.0.0.1:master.address}"
                        port="${2103:master.port}"/>
   <benchmark initSize="1" maxSize="${1:slaves}" increment="1">
      ...
      <TpccPopulation
         numWarehouses="1"
         cLastMask="0"
         olIdMask="0"
         cIdMask="0"
         populateOnOneCache="true"
         enablePopulate="true"/>

      <CsvReportGeneration/>
   </benchmark>

   ...

</bench-config>
```

At the end, to complete the population phase on all the four instances of the data platform (set $S1$), we execute again the runServer.sh command on the machines $IPslave2$, $IPslave3$, $IPslave4$. In this way the four instances automatically join the same cluster and they share the just populated initial data-set.

B. WPM execution

The WPM layer is used to monitor the data platform instances running on the set of machines $S1$ and to log the gathered statistics on machine $IPmaster$. For this reason we first run the WPM's Log Service on the machine $IPmaster$ by executing

```
$ cd wpm
$ ./run_log_service.sh
```

Then, on each machine in the set $S1$ we properly set the LogService_IP_Address attribute in the wpm/config/resource_consumer.config file (see Listing 3) to guarantee that WPM's Consumers are able to connect to the Log Service, and we run the pair Consumer/Producer in the following way:

```
$ cd wpm
$ ./run_consumer.sh
$ ./run_producer.sh
```

Listing 3: Consumer configuration

```
LogService_IP_Address=IPmaster
```

C. TPC-C benchmark execution

We change the benchmark.xml configuration file contained in RadargunClientServer/target/distribution/RadarGun-1.1.0-SNAPSHOT as shown in Listing 4, and we execute the following commands on machine $IPmaster$:

```
$ cd RadargunClientServer/target/distribution/RadarGun-1.1.0-SNAPSHOT
$ ./bin/benchmark.sh IPslave5 IPslave6 IPslave7 IPslave8
```

Listing 4: Benchmark configuration for TPC-C execution of four machines

```
<bench-config>
   <master bindAddress="${127.0.0.1:master.address}"
                port="${2103:master.port}"/>
   <benchmark initSize="4" maxSize="${4:slaves}" increment="1">
      ...
      <TpccPopulation
                numWarehouses="1"
                cLastMask="0"
                olIdMask="0"
                cIdMask="0"
                populateOnOneCache="false"
                enablePopulate="false"/>
      <TpccBenchmark
                numOfThreads="8"
                perThreadSimulTime="300"
                arrivalRate="0.0"
                backoffTime="1.0"
                thinkTime="1.0"
                paymentWeight="5.0"
                orderStatusWeight="70.0"/>

      <CsvReportGeneration/>
   </benchmark>
    ...
</bench-config>
```

In this way we will run four clients processes with eight benchmark threads per client, each one executing 70% of TPC-C Order-Status transactions, 5% of TPC-C Payment transactions and 25% of TPC-C New-Order transactions.

D. Actuator execution

Since the Actuator has to connect to the machines executing the data platform instances and it has to accept requests on a given ip:port end-point, we need to set the infinispanNodes, ip and port attributes in the AutonomicManagerActuator/actuator-server.properties file on the machine $IPmaster$ as shown in Listing 5

Listing 5: Actuator configuration

```
ip=IPmaster
port=8889
infinispanNodes=IPslave1 IPslave2 IPslave3 IPslave4
```

Afterwards we can run the Actuator on machine IPmaster as follows:

```
$ cd AutonomicManagerActuator
$ ./bin/runActuatorServer.sh
```

E. ANN execution

To train the neural network implemented within ANN we have to build an initial training set by monitoring preliminary runs of the benchmark and adopting a range of different configurations, e.g. number of data platform instances, number of client

threads, data replication degree. All the collected data have to be moved to a directory that will be accessed by the ANN module in order to execute the initial training phase and whose path has to be specified within the configuration file of the ANN module. In Listing 6, we show and example of configuration file for the ANN module, which is used for specifying the following set of parameters:

- $clientNormalization$: it is the maximum number of client that send requests to the system plus one.
- $serverNormalization$: it is the maximum number of available server plus one.
- $replicationNormalization$: it is the maximum level of replication that can be used inside the system (N.B serverNormalization <= replicationNormalization).
- $throughputNormalization$: it must be greater than the maximum throughput that can be obtained by the system.
- $responseNormalization$: it must be greater than the maximum response time that can be obtained by the system.
- $trainingFilesPath$: it is the relative path where the logging files used for the ANN training phase must be copied.
- $minimalReplication$: it is the minimal replication degree that we want to use inside the system (N.B. $minimalReplication < replicationNormalization$, $0 < minimalReplication \leq maximumNodeNumber$).
- $maximumNodeNumber$: it is the maximum number of server nodes that we want to use inside the system (N.B.: $maximumNodeNumber = serverNormalization - 1$).
- $optimizationTarget$: the values of this parameter have to be chosen in the domain $\{Throughput, Response\}$, where
  - $Throughput$ is specified if we want to maximize the system throughput;
  - $Response$ is specified if we want to minimize the response time.
- $monitoredNodes$: the list of server nodes that can be used by the system (the cardinality of this list must be equal to $maximumNodeNumber$'s value).
- $actuatorServer$: the ip address of the machine that hosts the Actuator server process.
- $actuatorServerPort$: the port number on which the Actuator server process is accepting connections.

Listing 6: ANN configuration

```
clientNormalization = 33
serverNormalization = 5
```

```
replicationNormalization = 5
throughputNormalization = 2000
responseNormalization = 17000
trainingFilesPath = ./trainingFiles
minimalReplication = 2
maximumNodeNumber = 4
optimizationTarget=Throughput
monitoredNodes =IPslave1 IPslave2 IPslave3 IPslave4
actuatorServer=IPmaster
actuatorServerPort=8889
```

After starting up the ANN module, it will automatically build the training set, it will train the neural network and will start to control the system. The number of nodes and the replication degree of the platform will be automatically controlled by the actuator process, which receives new predictions each time new monitoring samples are available and then, if needed, it triggers the commands aimed at changing the platform configuration. To run the ANN module we have to execute the following commands:

```
$ cd ANN
$ ./runANN.sh
```

Outcome

Figure 8 shows the system throughput, i.e. transactions committed per time unit, achieved by the Cloud-TM data platform during the execution of the TPC-C benchmark. In particular we run 32 clients threads on 4 different machines (i.e. IPslave5, IPslave6, IPslave7, IPslave8) that execute the TPC-C transactional logic on a cluster of 4 Cloud-TM data platform instances with data replication degree equals to 4 and under the ANN control. After 200 seconds since the start of the demo, the ANN component detects the current configuration to be sub-optimal and it triggers a configuration change to the Actuator process. In particular, ANN forecasts that the configuration with 2 data platform instances and data replication degree equals to 2 is optimal under the current workload and it forces that Actuator process to stop two of the running Cloud-TM instances and reduce the replication degree to 2. The result obtained after this reconfiguration is an increase by 43% of the overall system throughput.

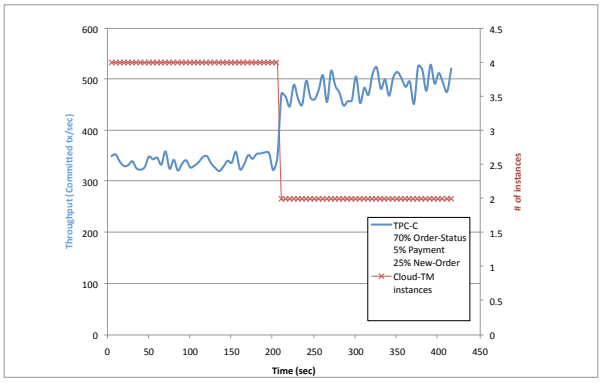

Figure 8: Execution of the example application under the ANN control.

6.1.2 Demo 2 - Automatic switching among replication protocols (based on MorphR)

In this section we discuss how to install and run an example application demonstrating the usage of MorphR when subjected to three very distinct workloads, showing that the system is able to recognize it is no longer using the optimal replication protocol and switch to the most appropriate one. We use the Radargun framework to generate these synthetic benchmarks, whose dynamics have already been explained in the beginning of Section 6.1.1.

The example starts with the system running a very low contention workload, especially suited for 2PC. Six minutes later, the workload is changed by increasing the contention, favouring the TOB protocol. Finally, in minute twelve we change the workload to the one with the highest contention, an optimal scenario for PB.

Like ANN, MorphR also relies on WPM to gather statistics and to be notified when new data is collected from all the nodes in the system. When MorphR receives new statistics, it queries the machine learning instance on whether the current replication protocol guarantees the highest throughput possible and, if not, switches the system's replication protocol to the optimal one. The machine learning instance relies on models built a priori; in this example the models were built by varying the parameters of the benchmark and running on a cluster with 10 nodes.

Configuring and running the example

As before, current example is already shipped in the VM image and it is ready to be executed after a brief configuration process. For self-containment, we list the steps needed to download, compile, configure and run the application.

The package can be downloaded at the following URL

```
https://github.com/cloudtm/cloudtm-autonomic-manager/
tree/master/demos/Demo2-MorphR
```

and it has the following structure

```
morphr
 |-MorphR/
 |-RadargunClientServer/
 |-run-test.sh
 |-environment.sh
 |-beforeBenchmark.sh
```

MorphR also requires the WPM runtime and a WPM Connector, both already covered in the previous example.

MorphR can be compiled by running the following commands:

```
$ cd MorphR
$ ant
```

Then RadargunClientServer can be compiled via maven by executing the following commands:

```
$ cd RadargunClientServer
$ mvn install
```

Before running this example you may need to edit the Morphr/buildPropertiesFile.sh configuration file, whose fields are listed below:

- namingHosts = the IPs of the slaves
- namingPort = the port where the JMX connection is open
- runDuration = the amount of time the MorphR should be active (in milliseconds)
- toQuery = if set to true the machine learner is queried, otherwise MorphR only prints information to build models
- outputFileName = file where the model is stored
- modelFilename = the name of the file with the model
- modelUsesTrees = model uses trees or rules
- forceStop = if set to false Stop and Go method is used, otherwise the fast switch is activated whenever possible
- abort = transactions are aborted during switches or not

The cluster parameter in the environment.sh must also be edited to include the names of the nodes in the system.

In order to run this example one simple has to execute the run-test.sh script, as it launches all the necessary components provided the directory has the following structure:

```
morphr
 |-MorphR/
 |-RadargunClientServer/
 |-wpm/
 |-Workload_Monitor_Connector/
 |-run-test.sh
 |-environment.sh
 |-beforeBenchmark.sh
```

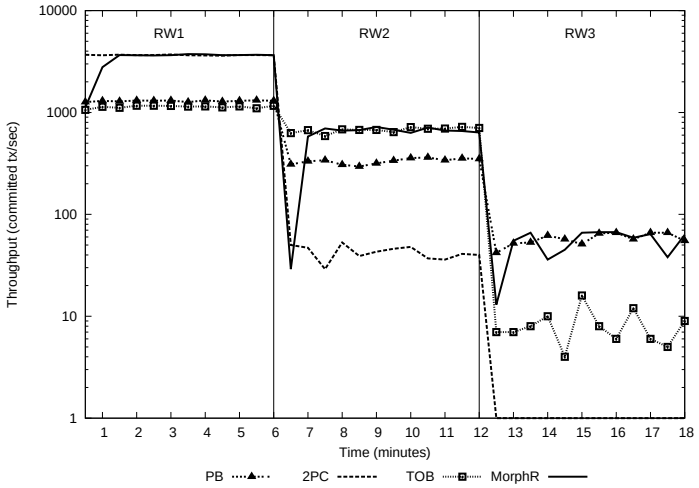

Figure 9: Comparision of the performance of MorphR with the static configurations.

Outcome

Figure 9 compares the throughput of MorphR with that of the three standalone replication protocols. This was obtained by executing the previously mentioned run.sh script. MorphR is able to recognise the changes in the workloads and switches the replication protocol to the optimal, based on the Machine learning output, which is queried every 70 seconds.

6.1.3 Demo 3 - What-if analysis (Based on TAS)

The purpose of the TAS' demo is to show the capabilities of the TAS' performance forecasting model. To this end, like for demo in Sec. 6.1.1 we will use the Radargun framework in order to run the TPC-C benchmark on top of Infinispan instances. Unlike demo in Sec. 6.1.1, however, the deployment of the application represents a use-case in which application tier and the data tier are collocated. This means that TPC-C transactions are generated directly on the Infinispan instances that will execute them. In this section we illustrate how to download, install and run the TAS' demo application, located in the `cloudtm-tas-demo` folder.

The VM image shipped with this document is already set up to run the application. However, we will describe all the necessary steps to perform a brand new installation on another machine.

System requirements The application requires the following packages to be installed

- Java 6 or higher
- Ant 1.8.4 or higher
- gnuplot
- git

Download and installation The package comes in the form of self-installer script that can be downloaded at the following URL

```
https://github.com/cloudtm/Demo3-TAS.git
```

and it has the following structure

```
TAS_Demo
 |-build.sh
```

To download and install the last version of the package, simply run

```
$ ./build.sh
```

Upon completion, the structure of the package is the following:

```
TAS_Demo
 |-ControllerTas
 |-RadargunTASDemo
 |-LatticeCloudTM
 |-wpm
 |-Workload_Monitor_Connector
```

A. Configuring and running the example

In this section, we will show how to configure and run the demo application.

Configuring wpm The configuration files relevant to wpm are in the wpm/config folder. The main parameters are already set up in order to match the requirements of the demo application. The only information that has to be provided at deployment time is the ip address of the machine hosting the LogService process. To this end, simply set

Listing 7: LogService configuration

```
...
LogService_IP_Address=IP_master
...
```

in the `log_service.config` file.

Configuring ControllerTas The configuration file of the ControllerTas is specified in the conf/ controller .xml file.

Listing 8: Controller configuration

```
<TasControllerConfiguration>
<ScaleConfig
     minNumNodes="2"
     maxNumNodes="10"
     minNumThreads="2"
     maxNumThreads="10"
     initNumNodes="2"
     initNumThreads="2"/>
<PlatformConfig
     numCores="8"/>
<GnuplotConfig
     exec="/usr/bin/gnuplot"/>
<DemoTransitoryConfig
    transtitoryTime="90"/>
</TasControllerConfiguration>
```

The semantic of these parameters is hereby explained

- minNumNodes, maxNumNodes, minNumThreads, maxNumThreads are the minimum and maximum values for nodes and threads considered in the what-if analysis
- initNumNodes, initNumThreads is the deployment configuration of the TPC-C benchmark, which must match the corresponding fields in the Radargun's configuration file.
- numCores in the number of cores per VM
- exec is the path to the gnuplot executable
- transitoryTime is the number of sec that the ControllerTas waits before considering stable the statistics collected from the slaves

The values for the numCores attribute has to be accordingly updated also in `conf/tas2.xml`.

Listing 9: TAS configuration

```
...
<PhysicalConfig
    numCores="8"/>
...
```

The other configuration parameters for the TAS module are not supposed to be modified, thus their meaning is not provided in this document.

Configuring Radargun The configuration parameters for Radargun are specified inRadargunTASDemo/conf/benchmark.xml are a subset of the ones already described in Sec. 6.1.1; therefore their explanation is omitted in this section.

Nevertheless, for the sake of clarity, here we remind that the following tags must match the values of the attributes initNumNodes and initNumThreads in the ControllerTas' configuration file.

Listing 10: Radargun configuration

```
<benchmark initSize="2"  maxSize="2" increment="1">
...
<TpccBenchmark
...
   numOfThreads="2"
...
/>
```

Please not that the values for initSize and maxSize have to coincide.

B. Running the application

To run the demo simply execute the following command on the master node.

```
$ ./run.sh "IP_slave1 IP_slave2 ... IP_slaveN"
```

This will start the LogService, the ControllerTas and the Radargun's master processes on the master node; at the same time, it starts the Radargun's slave processes and Wpm's producer and consumer processes in the slave nodes.

Periodically, the Wpm instances deployed over the slave nodes will push statistics to the LogService. Upon the receipt of these statistics, the LogService will trigger the what-if analysis process of the ControllerTas. Specifically, the ControllerTas will query TAS in order to obtain a forecast about the throughput that the application would deliver if deployed over different scales (in terms both of nodes in the platform and processing threads per node). The result of this analysis is stored both as a text file and as an image, stored respectively in the gnuplot/data and gnuplot/plot folders. Upon completion, a message is printed to notify the production of new results.

```
New plot produced and stored in gnuplot/plots/Throughput_1359906064803.eps
```

An example of produced plot is provided in Fig. 10

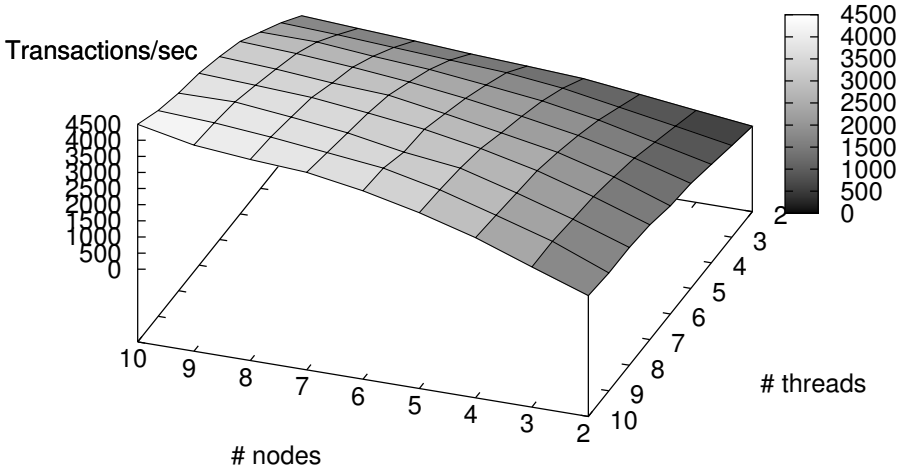

Figure 10: Outcome of TAS' what-if analysis

6.1.4 Demo 4 - QoS negotiation (based on DAGS)

In this example we demonstrate how the DAGS simulation framework can be exploited in order to support the QoS negotiation phase, and allow developers of Cloud-TM applications, and providers of the Cloud-TM platform (or of PaaS services embedding the Cloud-TM platform) to reach a mutual agreement on the SLAs, and corresponding costs, to be established.

To this end, the QoS/cost management framework of the Cloud-TM platform can rely on DAGS (or on other predictos, such as TAS) to predict main system performance indexes, including average transaction response time, throughput and transaction abort ratio. These indexes are predicted executing simulation runs while varying the number of clients, the number of nodes within the platform and the degree of data replication. The number of clients and the number of servers are varied within two intervals. Fixed the number of clients and servers, for each simulation run the degree of data replication is varied between 1 and the number of servers (thus including scenarios entailing both partial and full data replication). In this demo we exploit the application AutoDAGS, whose package is available at the following URL:

```
https://github.com/cloudtm/cloudtm-autonomic-manager/
tree/master/src/dags/AutoDAGS
```

AutoDAGS automatizes the whole simulation process by executing all simulation runs varying the above mentioned system configuration parameters, and providing final results in a single output file. The output file is compliant with the file format

required by the web application included in the QoS package allowing to visualize predicted system performance indexes with respect the SLA defined throught the WML templates. When executed, AutoDAGS performs the following computational steps:

- it parses the aggregated statistics included in the files provided by the WPM via the Log Service, in order to calculate the average computational resources demands of the different types of operations executed within the data platform (e.g. the average cpu service demand for get/put operations);
- it generates a configuration file containing all the configuration parameters needed for the simulation (also including the above-mentioned resource service demands);
- it executes e set of simulation runs varying the system configuration parameters, and eventually produces the output file.

Configuring and running the example

DAGS and AutoDAGS can be compiled using the GCC compiler version 4.4.3 or later, and using the makefiles included in both DAGS and AutoDAGS packages. To use AutoDAGS, the executable file generated by compiling DAGS must be placed within the same folder of the executable file generated by compiling AutoDAGS. AutoDAGS requires the following input parameters:

- m and M : (mandatory) they specify the minimum and the maximum, respectively, timestamp of data generated by the WPM to be parsed.
- f : (mandatory) it specifies the full path of the file (or the list of files) that contains data generated by the WPM to be parsed. Multiple files must be separated with a comma;
- s and S: they specify the minimum and the maximum, respectively, number of nodes to be used in the simulation .
- c and C: they specify the minimum and the maximum, respectively, number of clients to be used in the simulation (default 1).
- e: it specifies the number of transactions to be executed by each client in a simulation run (default: 10000).
- d: it specifies the number of the data objects to be used in the simulation (default 100000)

An example command line to execute AutoDAGS is:

```
$ ./AutoDAGS -m 1 -M 100000 -f data1.log,data2.log -s 1 \
     -S 4 -c 10 -C 20 -e 100000 -d 500000
```

where $data1.log$ and $data2.log$ are assumed to be files produced by the WPM.

At the end of the simulation process, the file output.txt will contain a set of rows (each one as a result of a simulation run), each one including (separated by comma):

- the number of clients used in the simulation run;
- the number of servers used in the simulation run;
- the number of clients used in the simulation run
- the predicted average transaction response time;
- the predicted transaction throughput;
- the predicted transaction abort ratio.

Figure 11 shows a graph depicting the average transaction response calculated using AutoDAGS. Predictions are calculated for a scenarions with 40 concurrent clients, varying the number of modes between 8 and 16, and, for a give number of server, varying the data replication degree between 1 and the number of server.

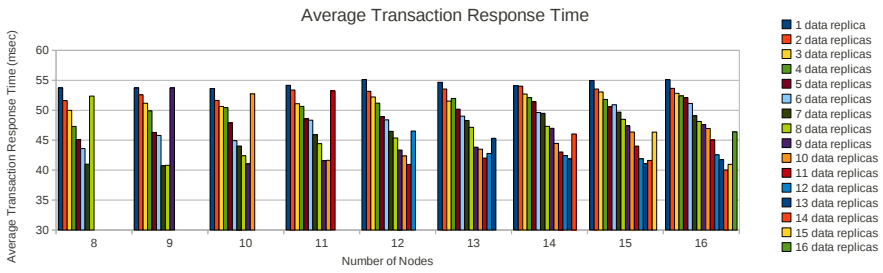

Figure 11: Example of the predicted average transaction response time

Figure 12 shows a print-screen of the web console supporting the SLA negotiation phase (see Section 7), and highlighting how the predictions generated by DAGS can be exploited, by both the platform provider and the application developer, in order to reach an agreement on the QoS levels to be guaranteed, as well as on the corresponding operational costs (computable based on the amount and type of computation resources necessary to ensure pre-determined QoS levels). Charts highlight using red color predicted values which violate the SLA defined by the user throught the WML templates. In the example depicted in the figure, system configurations which are expected to provide a throughput greater than 80 transaction per seconds are expected violate the average transaction response time as defined by user, while system configurations which are expected to provide a throughput greater than 90 transaction per seconds are expected to violate the maximum accettable abort rate.

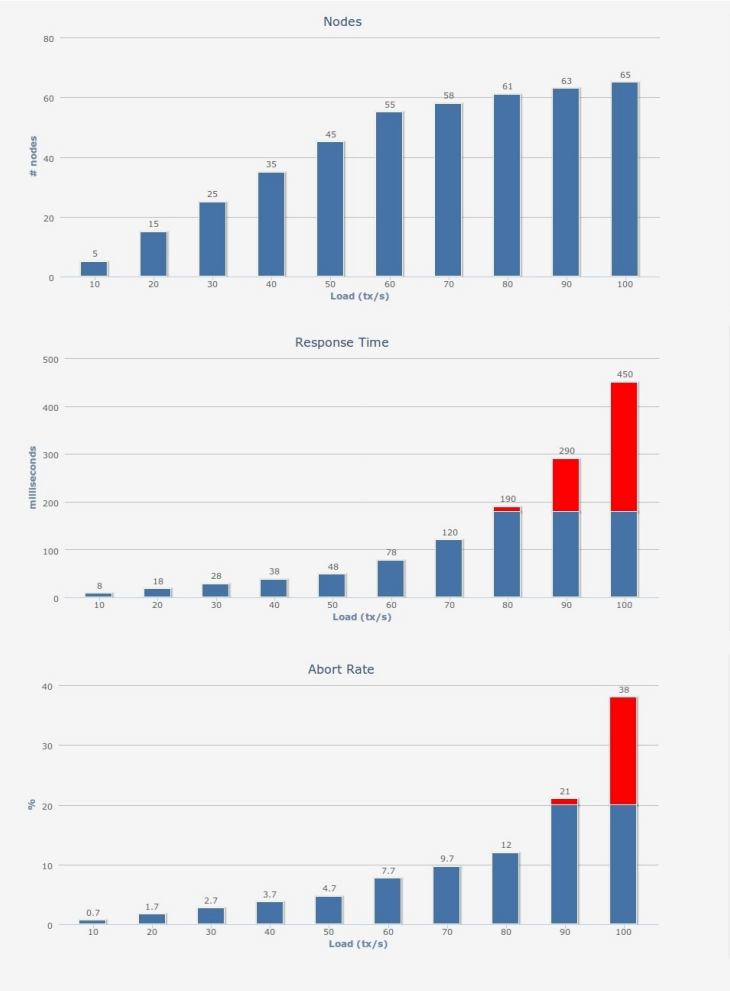

Figure 12: Example illustrating how the predictions generated by DAGS can be exploited in the context of the SLA definition phase.

6.1.5 Demo 5 - Self-tuning data placement (based on AUTOPLACER)

In this section we demonstrate the auto placer optimizer. For this demo, we use a synthetic benchmark named RadarGun, which was configured to achieve some data locality over the cluster. Each machine in the cluster performs a read dominant workload and it has a disjoint subset of the total number of keys that is most frequently accessed by each machine. In more detail, 90% of the time, each machine chooses a key from it local subset and the remaining for other machines' subsets.

The RadarGun is composed by a master node, referred to hereinafter as master_node, and by a set of slaves nodes, referred to hereinafter as slave_nodes. The master_node is responsible to coordinate all the benchmark phases (such as cluster creation, population, benchmark, etc.) which are executed by all the slave_nodes. All the steps described below must be performed in the master_node.

The VM image contains the following software package:

```
Demo5-AUTOPLACER
|-Csv-reporter/
|-Machine-learner/
|-Radargun/
|-www/
\-dist/
```

The dist/ provides the compiled code ready to be executed, after some configurations. However, it is possible to download the source code and compile it yourself.

Download and Compile the source code

All the source code is available on GitHub in the following url:

```
https://github.com/cloudtm/cloudtm-auto-placer
```

In order to download and compile the source code, perform the following steps:

```
$ git clone git://github.com/cloudtm/cloudtm-auto-placer.git
$ cd cloudtm-auto-placer
$ ./dist.sh
```

The ./dist.sh creates the folder dist/ similar to the one provided in the VM image. The next steps is to configure the demo.

Configure CSV Reporter

The CSV Reporter is a lightweight application that collects the Infinispan statistics over time and export them in a .csv file. In addition, we provide a PHP web page that analyses the .csv file generated and plots this information. The PHP web page is already installed in the VM image web server, so the next step is not necessary.

The PHP web page is *optional*, but if you want see the system's performance over time you have to copy the folder www/ for your web server folder (the web server installed must support PHP) and you should have write permissions in that folder. Assuming you are located in cloudtm−auto−placer folder and your web server folder is /var/www/html, perform the following steps

```
$ mkdir /var/www/html/example
$ # needed to save the .csv file generated
$ mkdir /var/www/html/example/current
$ cp -r www/* /var/www/html/example
$ chmod -R +w /var/www/html/example
```

The configuration for CSV Reporter needs a little changes. The file can be found in dist/conf/config.properties and the properties that may need to be modified are the following:

- reporter.ips: refers to a comma separated list of hostname:port of all the slave_nodes. The port value is the JMX port defined in dist/conf/environment.sh in property named JMX_SLAVES_PORT which value default to 9998;
- reporter.output_file: refers to the file path where the CSV Reporter will save the statistics. The default value is /var/www/html/example/current/reporter.css, but if you don't have the PHP web page, you can set the location to another folder, for example, /tmp/reporter.css;
- reporter.updateInterval: sets the update interval in seconds in which the CSV Reporter will collect the statistics. Each time the statistics are collected, a new line is added in the file set by reporter.output_file.

Configure Data Placement and Machine Learner

The data placement configuration can be found in the file dist/plugins/infinispan52cloudtm/conf/ispn.xml and the default configurations are the following: (note that the configurations does not need any changes when the VM image is used)

Listing 11: Data Placement configuration

```
<dataPlacement
   enabled="true"
   maxNumberOfKeysToRequest="1000"
   objectLookupFactory="org.infinispan.dataplacement.c50.
                                        C50MLObjectLookupFactory">
 <properties>
  <property
    name="keyFeatureManager"
    value="org.radargun.cachewrappers.RadargunKeyFeatureManager"
    />
  <property
    name="location"
    value="/tmp/ml"
    />
  <property
    name="bfFalsePositiveProb"
    value="0.001"
    />
  </properties>
</dataPlacement>
```

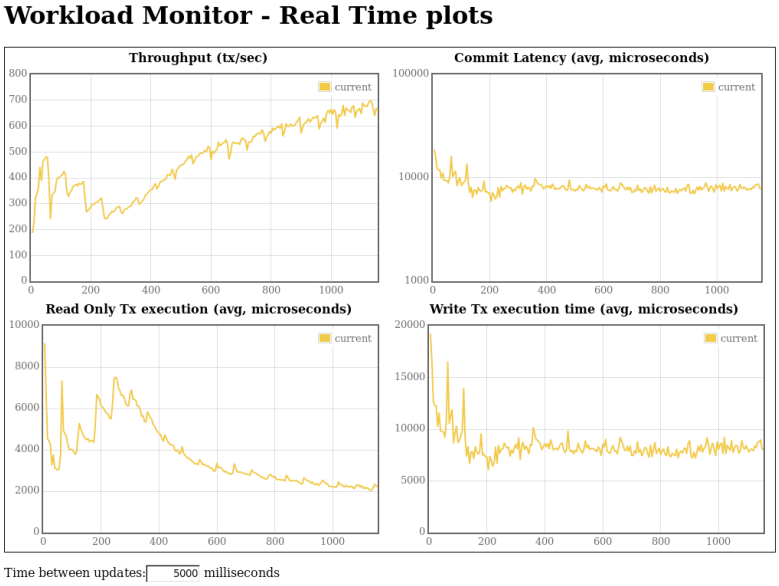

Figure 13: The VM ships with a lightweight statistics viewer that alows monitoring a set of key performance indicators, which allow monitoring the impact on performance of the self-tuning of the data placement in the Cloud-TM Data Platform.

The Machine Learner executable location should match with value defined in the location property. You may copy the folder in dist/ml to /tmp or change the property value to the new location. Note that the Machine Learner location must match in all the slave_nodes.

Before trying this demo, you need to check if the compiled Machine Learner runs in your system. If not, you can try to compile the C5.0[3] (the Machine Leaner used) to your system. If it is not possible, you can set the objectLookupFactory to the value org.infinispan.dataplacement.hm.HashMapObjectLookupFactory. This option will save the new key mapping in a Java Hash Map instead of the space efficient Machine Learner.

Running the Demo

After finishing all the previous steps, you only have to copy the dist/ folder to all the slave_nodes. After everything is copied, go to the master_node and perform the following command:

```
$ ./bin/benchmark.sh -i [number of nodes] [list of slaves_nodes]
```

Then you can check the system performance in the file set by reporter.output_file or in your browser in the link:

[3] http://www.rulequest.com/see5-info.html

```
http://master_node/example
```

The expected output is shown also in Figure 13

6.1.6 Demo 6 - Locality-aware Transaction Dispatching

In this section we discuss how to install and run an example application demonstrating the usage of the Locality-aware Transaction Dispatching mechanism by showing how it can be exploited in the context of a well-known benchmark for transactional applications, namely the TPC-W benchmark specified by the Transaction Processing Performance Council.

The benchmark implementation used in this demo has been obtained by porting a popular open-source Java implementation [2], which originally executed on top of Tomcat and MySQL, and adapting it to run on top of the Fénix Framework. This adaptation required transforming the hardcoded SQL queries into Java code. The detailed process is documented in [1].

The domain model

Using Fénix Framework, the TPC-W domain model is described using the *Domain Modeling Language (DML)*. DML is a domain-specific language that enables programmers to specify the structure of an application's domain model in a Java-like syntax, and then automatically generate the structural Java code for the classes. The advantage is that it is shorter to write, and it also represents relations between domain classes as a first-class concept. Furthermore, the DML compiler can generate backend-specific code that maintains the same programming API on top of the domain entities. This allows us to replace backends without having to change the application.

Installation

In the following we describe how to install and run the TPC-W Benchmark. This benchmark is composed by two separate components: the application server and the browser emulator. The former is deployed to a web server and the latter runs a given number of threads that simulate concurrent web clients interacting with the application.

Requirements

To run the benchmark:

- Apache Tomcat 6.x (should work on 7+ versions but hasn't been tested)
- JDK $\geq$ 1.5

To view the measurement results

- FreeMat

Get the source code

```
$ git clone git://github.com/fenix-framework/examples.git
```

In the examples/tpcw directory there are two independent programs in ‘server‘ and ‘client‘. The server is the web application to be deployed in Tomcat. The client contains the browser emulator. The client also contains some FreeMat files that process the benchmark's output, and the original TPC-W implementation for reference. From now on, operations regarding each program should be performed in that program's top directory (e.g., to perform the instructions regarding the server go to examples/tpcw/ server).

Running the benchmark

This is the *quick-and-dirty* how to get this benchmark running. For other alternatives please take a look at the configuration files within the source code.

Required configuration for Tomcat

In the server, edit the file build . properties and set the following properties to reflect your own Tomcat configuration:

Listing 12: Tomcat configuration

```
<!--  tomcat.username=test -->
<!--  tomcat.password=test -->
```

You should configure in your Tomcat a user that has permission to deploy a web application, and then take note of those credentials. Then start the Tomcat web application server.

Build and deploy the application server

To create the WAR packaged application do:

```
$ mvn package
```

You may change the backend to another by passing the property −Dfenixframework.code.generator in the previous command. Have a look at the pom.xml file for some possible values.‘

You may also change the following parameters when creating your application:

```
$ mvn package -DNUM_ITEMS=1000 -DNUM_EBS=10
```

The values shown are the default. They represent:

- NUM_ITEMS: the maximum number of books in the database and it must be one of 10^b with b in $[3; 7]$.
- NUM_EBS: the maximum number of emulated browsers that TPC-W will need to support. It must be greater than 0.

If the package commando succeeded you should be able to deploy the application with:

```
$ mvn tomcat6:deploy -Dtomcat.username=<username> -Dtomcat.password=<password>
```

Replace <username> and <password> with the correct credentials for your Tomcat installation.

NOTE: Be careful in changing the default values, as they have an implication in the number of instances of many domain objects and increasing them can easily grow the dataset out of manageable proportions! It is suggested that you start with the default values.

Also, if later you wish to replace the deployed application server with another version make sure you first remove the previously deployed code with:

```
$ mvn tomcat6:undeploy -Dtomcat.username=<username> \
  -Dtomcat.password=<password>
```

Check that the application is successfully deployed

Go to
`http://localhost:8080/tpcw/TPCW_home_interaction`
and you should see a page similar to Images in the loaded page do not show. This is not an error. It occurs simply because we didn't package the site's images in the web application.

You SHOULD NOT click any other links yet, because the data set is not populated**. This was just to check whether the application is successfully deployed.

Populate the dataset

Given that the default configuration uses Infinispan as an in-memory-only data repository, we need to generate a data set every time we redeploy the server application (which embeds Infinispan). To do so we've created a servlet that can be activated with:

```
$wget -q -T 0 -O - \
 "http://localhost:8080/tpcw/TPCW_populate?NUM_EBS=10&NUM_ITEMS=1000"
```

In here the values for NUM_EBS and NUM_ITEMS should always be less than or equal to the values used when building the application. Again, the parameters shown in the example are the default values, so they could be omitted.

If you do not have wget installed on your system you can alternatively access the given URL in your browser. Just make sure you **never** reload the page. It may take a while to populate the data. You must wait for it to finish before proceeding. The population servlet is meant to run only once and it does not support multiple executions.

Note: You may want to check in the server's logs \${CATALINA_HOME/logs/catalina.out) that the data population was successful.

Build the test client

Now move to the 'client' application and run:

```
$ ant dist-client-only
```

This creates the client application in the 'dist' directory along with a shell script to run it.

Run the client

Just do something like:

```
$ cd dist
$ chmod 700 rbe.sh
$ ./rbe.sh -r demo -u 60  -i 120 -d 20 -t 1 -b 1000 -n 3 -tt 0
```

This simulates 3 store clients continuously accessing the store during 120 seconds, with a previous warm-up period of 60 seconds and a ramp-down period of 20 seconds, using workload 1 (TPC-W's browsing mix with 5% write transactions). To better understand the parameters and the values that can be used run:

```
$ ./rbe.sh -h
```

Just take note that the value of flag -b must match the value ofNUM_ITEMS used previously.

Analyze the results

After the script completes a FreeMat file is produced with a name like rundemo_t1_e3_b1000_20111011_2301.m. This file must be processed in FreeMat. To do so you'll need the files in directory freemat. Just copy the generated file over tpcw.m and then open FreeMat.

```
$ cp rundemo_t1_e3_b1000_20111011_2301.m ../freemat/tpcw.m
```

In FreeMat you will probably need to configure its path to point to the freemat directory in the client. Running the function wips(tpcw) will produce a plot with the Web Interactions Per Second (WIPS) corresponding to the current contents of tpcw.m. The plot shows a timeline and the WIPS for every second of the execution. It also plots a curve with the average WIPS for the 30 seconds around a given point, and a line with the overall average WIPS.

Configuring the load balancer

Let us now describe how to setup and deploy a multiple node/instance version of the TPC-W benchmark with a load-balancer front-end.

We focus here on the information necessary for configuring the load-balancer. For information regarding the client/server please consult the appropriate documentation.

The current implementation of the TPC-W load-balancer supports 3 and 4 server node deployment configurations as well as 3 alternative request distribution policies, namely:

- RoundRobin - classic round-robin request distribution policy.
- LDA - request distribution policy that seeks to maximize data locality by clustering application data and functionality. In practice, it makes sure that every available server node is responsible for processing request types (services) which belong to a particular sub-group of all available requests types. The contents of the clusters-of-functionality-types are established through the use of the current state-of-the-art in clustering algorithms Latent Dirichlet Allocation (LDA). This policy only focuses on maximizing data locality and improving the efficiency of the target application in terms of usage of computational resources (mainly cpu and memory). It does not make any effort to distribute incoming requests in a way that leads to a uniform load distribution among existing nodes.
- LARD - a theoretically idealized request distribution policy that makes use of perfect a-priori knowledge about the target application behaviour. This policy incorporates knowledge covering the average time necessary for processing all request types, as well as the relative proportions in their invocations at run-time. By using this information, the policy makes an attempt to obtain a more uniform load-distribution among existing nodes, while, similarly to LDA, keeping each server node responsible for processing only a particular sub-group of requests types. It should be noted that the LARD policy does not take into account the actual domain data necessary for the execution of requests, when establishing the contents of the groups of requests that each server node should be responsible for. Last but not least, this policy was created for the purpose of comparing it against the developed LDA-based policy. The LARD policy is not something that can be realistically achieved in practice.

For more information about the policies, their effects and comparative results, please consult [12]

There are 2 points in the load balancer that need to be configured to have it operating properly.

The first of these is to ensure that the lb.conf configuration file is available in the classpath of the application. If you have trouble doing this, then edit the

tpcw/loadbalancer/src/java/pt/ist/fenixframework/example/tpcw/loadbalance/LoadBalance.java

and uncomment the

```
//conf = LoadBalanceConfig.loadConfig((new
         File("/path_to_tpcw/loadbalancer/conf/lb.conf")).toURI().toURL())
```

making sure that you indicate the proper path to the configuration file.

An example of the contents of a configuration file, for the RoundRobin policy, with 3 server nodes follows:

```
policy_class=pt.ist.fenixframework.example.tpcw.loadbalance.RoundRobinPolicy
number_of_replicas=3
replica1=http://node02:18080/tpcw
replica2=http://node03:18080/tpcw
replica3=http://node04:18080/tpcw
```

Configuration files, for the LARD, LDA and RoundRobin policies, for 3 and 4 server nodes deployment can be found in the tpcw/loadbalancer/conf folder:

```
tpcw/loadbalancer/conf
 |-LARDPolicy_3replica.conf
 |-LARDPolicy_4replica.conf
 |-LDAPolicy_3replica.conf
 |-LDAPolicy_4replica.conf
 |-RoundRobinPolicy_3replica.conf
 |-RoundRobinPolicy_4replica.conf
```

The only changes necessary to be done on these files would be to change the information about the correct URL where the TPC-W replicas have been deployed. Depending on the policy with which the load-balancer should operate, the appropriate configuration file should be renamed to lb.conf and added to the application classpath.

The second aspect that needs to be configured (only for the LDA and LARD policies) is the path to the serialized version of the contents of the groups of request types that should be processed by each server node. This can be accomplished by editting the localPath variable in

tpcw/ loadbalancer / src / java / pt / ist /fenixframework/example/tpcw/loadbalance /LDAPolicy.java or tpcw/ loadbalancer / src / java / pt / ist /fenixframework/example/tpcw/loadbalance /LARDPolicy.java

to point to the correct path where this data is available. The folders for the LARD policy are:

```
tpcw/loadbalancer/data_clustering/results/3clusters_lard/
tpcw/loadbalancer/data_clustering/results/4clusters_lard/
```

while for the LDA policy are:

```
tpcw/loadbalancer/data_clustering/results/3clusters_lda/
tpcw/loadbalancer/data_clustering/results/4clusters_lda/
```

Last but not least, after having deployed the TPC-W server nodes and the load-balancer, make sure that the workload generator (in the client) is configured to send all requests to the URL where the load-balancer has been deployed.

References

[1] http://www.esw.inesc-id.pt/permalinks/fenix-framework-tpcw

[2] http://tpcw.deadpixel.de/

[3] A. Bryson and Y. Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Halsted Press book'. Taylor & Francis, 1975.

[4] M. Couceiro, P. Romano and L. Rodrigues Chasing the Optimum in Replicated In-memory Transactional Platforms via Protocol Adaptation *Technical Report, INESC-ID, Nov. 2012*

[5] P. Di Sanzo, F. Antonacci, B. Ciciani, R. Palmieri, A. Pellegrini, S. Peluso, F. Quaglia, D. Rughetti, and R. Vitali. A Framework for High Performance Simulation of Transactional Data Grid Platforms. In *6th International ICST Conference on Simulation Tools and Techniques (SIMUTools)*, 2013.

[6] P. Di Sanzo, D. Rughetti, B. Ciciani, and F. Quaglia. Auto-tuning of cloud-based in-memory transactional data grids via machine learning. In *Proc. 2nd IEEE International Symposium on Network Cloud Computing and Applications (NCCA)*, NCCA '12. IEEE Computer Society, 2012.

[7] D. Didona, P. Romano, S. Peluso, and F. Quaglia. Transactional auto scaler: elastic scaling of in-memory transactional data grids. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 125–134, New York, NY, USA, 2012. ACM.

[8] Garbatov, S., Cachopo, J. and Pereira, J. Data Access Pattern Analysis based on Bayesian Updating *Proceedings of the First Symposium of Informatics* (INForum 2009), Lisbon.

[9] Garbatov, S. and Cachopo J. Data Access Pattern Analysis and Prediction for Object-Oriented Applications *INFOCOMP Journal of Computer Science* 10(4): 1-14.

[10] Garbatov, S. and Cachopo J. Predicting Data Access Patterns in Object-Oriented Applications Based on Markov Chains *Proceedings of the Fifth International Conference on Software Engineering Advances* (ICSEA 2010), Nice, France.

[11] Garbatov, S. and Cachopo J. Importance Analysis for Predicting Data Access Behaviour in Object-Oriented Applications *Journal of Computer Science and Technologies* 14(1): 37-43.

[12] Garbatov, S. and Cachopo J. Explicit use of working-set correlation for load-balancing in clustered web servers *Proceedings of the Seventh International Conference on Software Engineering Advances* (ICSEA 2012), Lisbon, Portugal.

[13] Hoffman, Matthew D., David M. Blei, and Francis Bach. Online learning for latent dirichlet allocation. *Advances in Neural Information Processing Systems 23* (2010): 856-864.

[14] HPDCS Research Group. ROOT-Sim: The ROme OpTimistic Simulator - v 1.0. `http://www.dis.uniroma1.it/~hpdcs/ROOT-Sim/`.

[15] JBoss Infinispan. Infinispan Cache Mode. https://docs.jboss.org/author/display/ISPN/Clustering+modes, 2011.

[16] A. Metwally, D. Agrawal, and A.El Abbadi.

[17] J. Paiva, P. Ruivo, P. Romano and L. Rodrigues AUTOPLACER: scalable self-tuning data placement in distributed key-value stores *Technical Report, INESC-ID, Nov. 2012*

[18] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, and L. Rodrigues. When scalability meets consistency: Genuine multiversion update-serializable partial data replication. *2012 IEEE 32nd International Conference on Distributed Computing Systems*, 0:455–465, 2012.

[19] J.R. Quinlan. C5.0 `http://www.rulequest.com/see5-info.html`.

[20] J.R. Quinlan. Cubist. `http://www.rulequest.com/cubist-info.html`.

[21] D. E. Rumelhart and R. J. W. Geoffrey E. Hinton. Learning representations by back-propagating errors. *Nature*, 323.

[22] S. J. Russell, P. Norvig, J. F. Candy, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[23] TPC Council. TPC-C Benchmark, Revision 5.11. Feb. 2010.