ARTICLE IN PRESS

Simulation Modelling Practice and Theory xxx (2015) xxx–xxx

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

A flexible framework for accurate simulation of cloud in-memory data stores

P. Di Sanzo [a], F. Quaglia [a,*], B. Ciciani [a], A. Pellegrini [a], D. Didona [b], P. Romano [b], R. Palmieri [c], S. Peluso [c]

[a] DIAG, Sapienza Universita' di Roma, Via Ariosto 25, 00185 Roma, Italy
[b] INESC-ID, R. Alves Redol 9, 1000-029 Lisbon, Portugal
[c] Virginia Tech, Blacksburg, VA 24061, United States

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Cloud data stores
In-memory data stores
What-if analysis
Simulation/machine-learning integration

ABSTRACT

In-memory (transactional) data stores, also referred to as data grids, are recognized as a first-class data management technology for cloud platforms, thanks to their ability to match the elasticity requirements imposed by the pay-as-you-go cost model. On the other hand, determining how performance and reliability/availability of these systems vary as a function of configuration parameters, such as the amount of cache servers to be deployed, and the degree of in-memory replication of slices of data, is far from being a trivial task. Yet, it is an essential aspect of the provisioning process of cloud platforms, given that it has an impact on the amount of cloud resources that are planned for usage. To cope with the issue of predicting/analysing the behavior of different configurations of cloud in-memory data stores, in this article we present a flexible simulation framework offering skeleton simulation models that can be easily specialized in order to capture the dynamics of diverse data grid systems, such as those related to the specific (distributed) protocol used to provide data consistency and/or transactional guarantees. Besides its flexibility, another peculiar aspect of the framework lies in that it integrates simulation and machine-learning (black-box) techniques, the latter being used to capture the dynamics of the data-exchange layer (e.g. the message passing layer) across the cache servers. This is a relevant aspect when considering that the actual data-transport/networking infrastructure on top of which the data grid is deployed might be unknown, hence being not feasible to be modeled via white-box (namely purely simulative) approaches. We also provide an extended experimental study aimed at validating instances of simulation models supported by our framework against execution dynamics of real data grid systems deployed on top of either private or public cloud infrastructures. Particularly, our validation test-bed has been based on an industrial-grade open-source data grid, namely Infinispan by JBoss/Red-Hat, and a de-facto standard benchmark for NoSQL platforms, namely YCSB by Yahoo. The validation study has been conducted by relying on both public and private cloud systems, scaling the underlying infrastructure up to 100 (resp. 140) Virtual Machines for the public (resp. private) cloud case. Further, we provide some experimental data related to a scenario where our framework is used for on-line capacity planning and reconfiguration of the data grid system.

© 2015 Elsevier B.V. All rights reserved.

* Corresponding author.
E-mail address: quaglia@dis.uniroma1.it (F. Quaglia).

http://dx.doi.org/10.1016/j.simpat.2015.05.011
1569-190X/© 2015 Elsevier B.V. All rights reserved.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

1. Introduction

The advent of cloud computing has led to the proliferation of a new generation of in-memory, transactional data platforms, often referred to as NoSQL data grids, among which we can find products such as Red Hat's Infinispan [1], VMware vFabric GemFire [2], Oracle Coherence [3] and Apache Cassandra [4]. These platforms well meet the elasticity requirements imposed by the pay-as-you-go cost model since they (a) rely on a simplified key-value data model (as opposed to the traditional relational model), (b) employ efficient in-memory replication mechanisms to achieve data durability (as opposed to disk-based logging) and (c) natively offer facilities for dynamically resizing the amount of hosts within the platform. They are therefore widely recognized as a core technology for, e.g., emerging big data applications to be hosted in the cloud.

However, beyond the simplicity in their deploy and use, one aspect that still represents a core issue to cope with when adopting in-memory NoSQL data grids is related to the (dynamic) resize and configuration of the system. This is of paramount importance in the cloud anytime some predetermined Service Level Agreement (SLA) needs to be matched while also minimizing operating costs related to, e.g., renting the underlying virtualized infrastructure. However, accomplishing this goal is far from being trivial, as forecasting the scalability trends of real-life, complex applications deployed on distributed in-memory transactional platforms is very challenging. In fact, as recently shown in [5], when the number of nodes in the system grows and/or the workload intensity/profile changes, the performance of these platforms may exhibit strong non-linear behaviors, which are imputable to the simultaneous, and often inter-dependent, effects of contention affecting both physical (CPU, memory, network) and logical (conflicting data accesses by concurrent transactions) resources.

Recent approaches have tackled the issue of predicting the performance of these in-memory data grid platforms (e.g. to assist dynamic reconfiguration processes) by relying on analytical modeling, machine learning or a combination of the two approaches (see, e.g., [7,5]). In this article we provide an orthogonal solution which is based on the combination of discrete event simulation and machine learning techniques. Specifically, we present a framework for instantiating discrete event models of data grid platforms, which can be exploited for what-if analysis in order to determine what would be the effects of reconfiguring various parameters, like: (i) the number of cache servers within the platform; (ii) the degree of replication of the data-objects; and (iii) the placement of data-copies across the platform. Hence, it can be used to determine well-suited configurations (e.g. minimizing the cost for the underlying virtualized infrastructure) vs variations of the volume of client requests, the actual data conflict and the locality of data accesses. It can also be used for long term SLA-driven planning in order to determine whether the data grid can sustain an increase in the load volume and at what operational cost – as a reflection of the increased amount of resources that shall be provisioned from the cloud infrastructures.

The framework has been developed as a C static library implementing data grid models developed according to the traditional event-driven simulative approach, where the evolution of each individual entity to be simulated within the model is expressed by a specific event-handler.[1] On the other hand, the library has been structured in order to allow easy development of models of data grid systems offering specific facilities and supporting specific data management algorithms (e.g. for ensuring consistency of replicated data). As for this aspect, distributed data grids relying on two-phase-commit (2PC) as the native scheme for cache server coordination, as typical of most of the mainstream implementations (see, e.g., [1]), have an execution pattern already captured by the skeleton model offered by the library. Hence, models of differentiated 2PC-based data management protocols could be easily implemented on top of the framework. Further, models natively offered within the framework include those of data grids ensuring repeatable read semantics, which are based on lazy locking. Models of primary data ownership vs multi-master schemes are also natively supported.

The ability of our simulation framework to reliably capture the dynamics of data grid systems is strengthened by the combination of the white-box simulative approach with black-box machine learning techniques. The latter have been demonstrated to represent an essential support for coping with non-linearity and for complementing white-box approaches (e.g. via ensemble schemes) especially when predicting performance with specific configurations of the system (or workload) parameters [6].

In our framework, the usage of a black-box approach aims to capture (and to predict) the data-transport/networking sub-system dynamics (as observable from outside of such sub-system). This spares users from the burden of explicitly modeling the internal structure and behavior of the network layer within the simulation code, which is known to be an error-prone task given the complexity and heterogeneity of existing network architectures and/or message-passing/group-communication systems [8].[2] Also, the reliance on machine learning for modeling network dynamics widens the framework practical usability in modeling data grid systems deployed over virtualized cloud environments where users have little or no knowledge of the underlying network topology/infrastructure and of how the lower level message passing sub-systems are structured. For these scenarios, the construction of white-box simulative models would not only be a complex task, rather it would be unfeasible.

We also present a case study, used as a support for the validity of the proposed modeling approach, where we compare simulation outputs with measurements obtained running the YCSB benchmark by Yahoo [10], in different configurations, on top of the Infinispan data grid system by JBoss/Red-Hat [1], namely one of the mainstream data layers for the JBoss

[1] The actual code implementing the framework is freely available for download at the URL http://www.dis.uniroma1.it/~hpdcs/software/dags-with-cubist.tar.

[2] Group communication systems such as [9] are often used as data exchange layers within real data grid products. They typically exhibit complex dynamics that can vary on the basis of several parameters, hence being difficult to be reliably captured via white-box models.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

application server. We note that the YCSB benchmark has been designed to explicitly assess the run-time behavior of cloud data stores, and has been already exploited as a reference in a set of recent studies (see, e.g., [5]), hence looking as an ideal candidate for our case study. Also, Infinispan supports distributed data management schemes that can be considered as instances of "archetypal" ones, which strengths the relevance of our case study in assessing the actual quality of the models that can be instantiated via the framework. Further, the experiments have been conducted by relying on both private and public (namely FutureGrid [35]) cloud systems, by scaling the underlying infrastructure up to 140 Virtual Machines for the private cloud, and up to 100 Virtual Machines for the public one. By the validation study, the framework provides (at least) 80% accuracy in predicting core performance metrics such as the system throughput across all the tested configurations, and on the order of 95% accuracy for most of them.

Beyond reporting validation data, we also provide experimental results related to a scenario where the framework is used for on-line capacity planning and reconfiguration of the data grid system. This part of our experimental analysis still relies on Infinispan as the data grid system, this time deployed on a virtualized platform supported by Amazon EC2 [36]. Further, the overall description of the experimental settings we adopted allows for providing information on how the framework can be integrated (e.g. for capacity planning usage) within an operational data grid environment.

The remainder of this article is structured as follows. In Section 2 we discuss related work. The framework organization is presented in Section 3. Experimental data are reported in Section 4.

2. Related work

The issue of studying/predicting the performance of data grids has been addressed in literature according to differentiated methodologies. The recent works in [5,11,12] provide approaches where analytical modeling and machine learning are jointly exploited in the context of performance prediction of data grid systems hosted on top of cloud-based infrastructures. The analytic part is mainly focused to capturing dynamics related to the specific concurrency control algorithm adopted by the data grid system, while machine learning is targeted at capturing contention effects on infrastructure-level resources. Differently from our approach, these works cope with specific data grid configurations (e.g. specific data management algorithms and/or specific workload profiles) to which the analytical models are targeted. For example, they assume arrivals of transactions to the system to form a Poisson process. However, recent works suggest that, in large scale data centers, the inter-arrival time of requests to a data grid may not follow the exponential distribution [13]. In the same guise, those models are bound to specific data access pattern dynamics (e.g. in terms of data locality), which are not general enough to encompass complex data-partitioning schemes across the servers [14]. Instead, we offer a framework allowing the user to flexibly model, e.g., differentiated data management schemes without imposing specific assumptions on the workload and data access profile (in fact real execution traces can be used to drive the simulated data access).

The proposals in [15,16] are based on the exclusive usage of machine learning, hence they provide performance prediction tools that do not have the capability to support what-if analysis in the wide (e.g. by studying the effects of – significant – workload shifts outside the workload-domain used during the machine learning training phase). Rather, once a machine learning-based model is instantiated via these tools, it stays bound to a specific scenario (e.g. to a specific deploy onto a given infrastructure), and can only be used to (dynamically) reconfigure the target data grid that has been modeled. We retain similar capabilities; however, by limiting the usage of the machine learning component to predicting messaging/networking dynamics, we also offer the possibility to perform what-if analysis and exploration of non-instantiated configurations (e.g. in terms of both system setting and workload profile/intensity).

One approach close to our proposal can be found in [17]. This work presents a simulation layer entailing the capabilities of simulating data grid systems. Differently from this approach, which is purely simulative, our proposal exhibits higher flexibility in terms of its ability to reliably model the dynamics of data grid systems in the cloud thanks to the combination of simulative and machine learning techniques. In fact, as already pointed out, the machine learning part allows for employing the framework in scenarios where no (detailed) knowledge on the structure/internals of the networking/messaging system to be modeled is (or can be) provided. As for this aspect, the usage of machine learning for the performance prediction of group communication systems has been pioneered in [8]. However, the idea of combining simulative and machine learning-based models is, to the best of our knowledge, still unexplored in the literature.

Simulation of data grid systems has also been addressed in [18]. In this proposal, the modeling scheme of the data grid is based on Petri nets, which are then solved via simulation. With respect to this solution, we propose a functional model that does not explicitly rely on modeling formalisms, except for the case of the CPU, which is modeled via queuing approaches. Further, one relevant difference between the work in [18] and our proposal lies in that our simulation models are able to simulate complex transactional interactions entailing multiple read/write (namely get/put) operations within a same transaction. Instead, the work in [18] only models single get/put interactions to be issued by the clients.

Still related to our proposal are the simulation models developed in [19]. However, unlike this article, the focus of that work is on modelling lower levels dynamics related to IaaS management (e.g. scheduling of Virtual Machines to a set of physical resources). Finally, a work still related to our proposal, although marginally, can be found in [20], where a simulation environment for backup data storage systems in peer-to-peer networks is presented. Compared to our proposal, this work is focused on lower level data management aspects, such as the explicit modeling of actual stable storage devices. Instead, our focus is on distributed dynamics at the level of in-memory data storing systems, which are essentially independent of (and orthogonal to) those typical of stable storage technologies.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

3. The framework

The data grid architectures we target in our framework entail two types of entities, namely:

- *cache servers*, which are in charge of maintaining copies of entire, or partial, data-sets;
- *clients*, which issue transactional data accesses and/or updates towards the cache servers.

The cache servers can be configured to run different distributed protocols in order to guarantee specific levels of isolation and data consistency while supporting transactional data accesses. For instance, the 2PC protocol can be exploited in order to guarantee atomicity while updating distributed replicas of the same data-object, as it typically occurs in commercial in-memory data platform implementations (see, e.g., [1]). Also, an individual transactional interaction issued by any client can be mapped onto either a single put/get operation of a data-object, or a more complex transactional manipulation involving several put/get operations on multiple data-objects, which is demarcated via begin and end statements.

In the next subsections we initially focus on the structure and discrete-event patterns of cache server and client simulation objects. Successively, we enter the details of the machine learning approach used to model message delivery latencies across the system components, and of its integration with the simulative part of the framework.

3.1. The cache server simulation-object

A cache sever simulation object can be schematized as shown in Fig. 1. By the scheme we can identify four main software components:

- the transaction manager (TM);
- the distribution manager (DM);
- the concurrency control (CC); and
- the CPU.

Any simulation event destined to the cache server is eventually passed as input to TM, which acts therefore as a front-end for event processing. Upon the scheduling of any event, TM determines the amount of time needed to process the requested operation, which depends on the type of the scheduled event, and on the current CPU load. Then, the CPU load is updated on the basis of the requested operation and a CPU_complete event is scheduled at the proper simulation time.

To determine the CPU processing delay, the CPU has been modeled as a G/M/K queue, which allows capturing scenarios entailing multiple CPU-cores. Although more sophisticated models could be employed (see, e.g., [21]), we relied on G/M/K queues since, in our target simulation scenarios, the core dynamics of interest are the ones related to contention on logical resources, namely data-objects, rather than physical resources, and to distributed (locking) strategies for the management of atomicity of the updates of distributed/replicated data copies. Hence, distributed coordination delays play a major role in the determination of the achievable performance, as compared to CPU delays for processing local operations. Consequently, the G/M/K queue is expected to be a fairly adequate model for the objectives of the framework. Also, given that in conventional operating systems swap operations of data that are out of the working set are typically executed out of the critical path of the CPU processing activities (e.g. by relying on demons, such as `kswapd` in Linux systems) the effects of virtual memory on the latency of operations provided within the data grid simulation model are not explicitly modeled.[3]

When a local processing operation is completed, TM takes again control (via the aforementioned CPU_complete event) and updates the cache server simulation state depending on the operation type.

As for events scheduled by client simulation objects towards the cache servers, the corresponding event-types within the framework skeleton are listed below:

- begin, used to notify TM that a new transactional interaction has been issued by some client, which must be processed by the cache server;
- get, used to notify that a read operation on some data-object has been issued by the client within a transaction;
- put, used to notify that a write operation on some data-object has been issued by the client within a transaction; and
- commit, used to indicate that the client ended issuing operations within a transaction, whose commit can therefore be attempted.

[3] Virtual memory operations typically occur along the critical path in case of access to empty-zero memory, the so called minor faults, or when a change of locality towards data previously swapped-out is experienced, the so called major faults. For minor faults, no I/O operation is requested, hence these exhibit very limited overhead for their management in CPU, neglecting which in the simulative model is expected not to significantly impact model fidelity. On the other hand, classical locality principles lead the software to infrequently request the access to data that have gone out of the current working set, which makes the event of major faults statistically less relevant.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

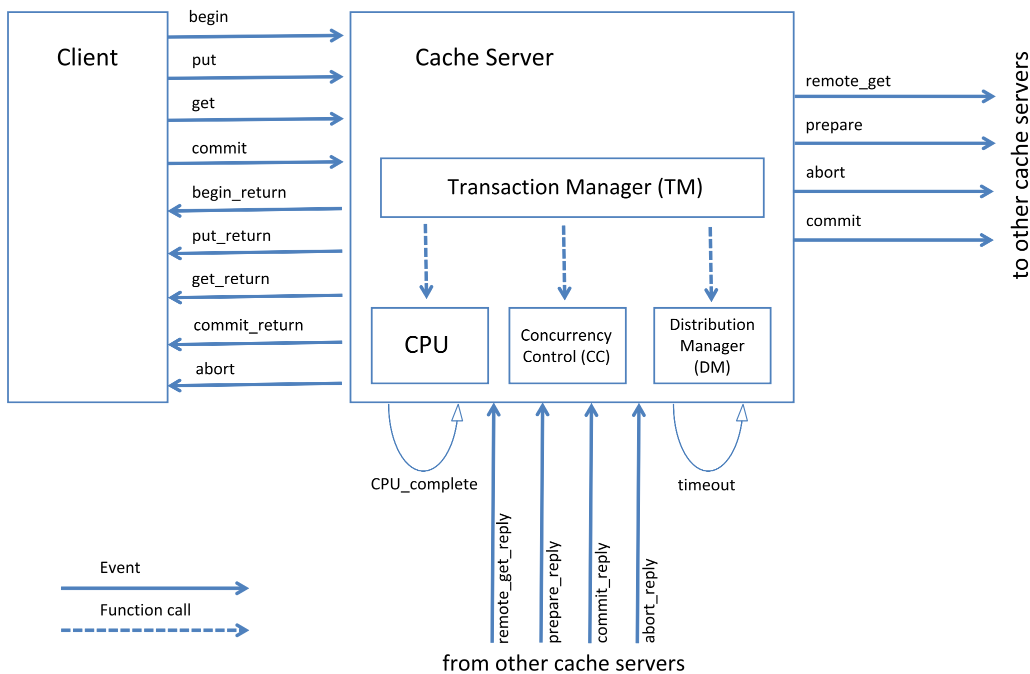

Fig. 1. Client and cache server simulation objects.

The handling of each type of event is explained in the following sections.

3.1.1. The begin *event*

In Fig. 2 we provide a sequence diagram showing how the begin event is handled. As illustrated, the interactions between client and cache server simulation objects ar asynchronous, given that they are based on the exchange of timestamped events.

The actual processing activities for the begin event at the cache server side (which are related to the red-box in the diagram) are performed by the function `setupTransaction`, which simply takes as input the current simulation time and pointers to two records of type `TxInfo` and `TxStatistics`, which are automatically allocated by the cache server, whose structure can be defined by the simulation modeler.[4]

The reason for allowing the modeler to exploit two different data types lies in that the content of `TxInfo` is made valid across cache servers. In fact, it is automatically transferred to remote cache server simulation objects when cross scheduling of events is actuated. This is relevant in any simulated scenario where some transaction set-up (or transaction state) information needs to be made available to remote cache servers, e.g., for distributed contention management purposes. On the other hand, the content of `TxStatistics` is not transferred across different simulation objects, being it locally handled by the cache server acting as the coordinator of the transaction.

3.1.2. get *and* put *events*

In Fig. 3 we show the sequence diagram illustrating the handling of get simulation events, which cause the TM module to query (via synchronous procedure invocation) the DM module. This is done in order to get information about what cache servers figure as the owners of the data-object to be accessed. In our architecture, the DM module provides this information back in the form of a pointer to a list of cache server identifiers (hence simulation object identifiers), where each record also keeps additional information specifying whether a given cache sever is (or is not) the primary owner of a copy of the data-object to be accessed.

Then, the cache sever initially determines whether it is the owner of a copy of the data-object. In the positive case, the read operation on the data-object will simply result in an invocation of the CC module on this same cache server instance. Otherwise, remote_get simulation events are scheduled for all the cache servers figuring as owners of a copy of the data-object.

One important aspect associated with the above scheme is that the get operation may be blocked at the level of CC, depending on the actual policy for controlling concurrency. On the other hand, even in case of CC simulated algorithms

[4] The only constraint is that the top standing field of `TxInfo`, must be of type `TxId`, which keeps the transaction unique identifier, automatically generated by the cache server just to facilitate the actual management within model execution.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

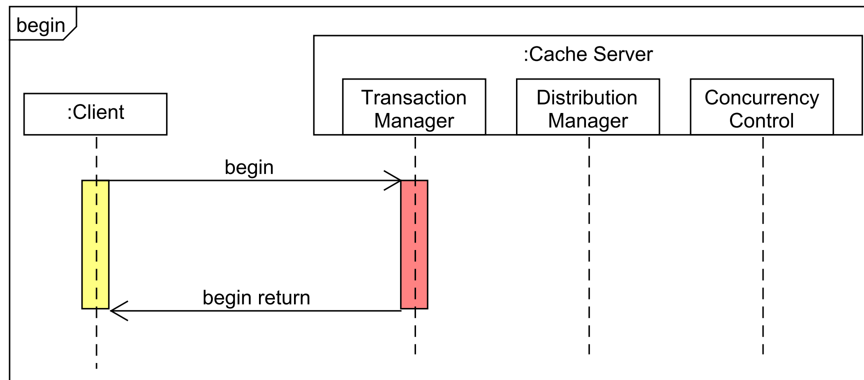

Fig. 2. Management of the begin event.

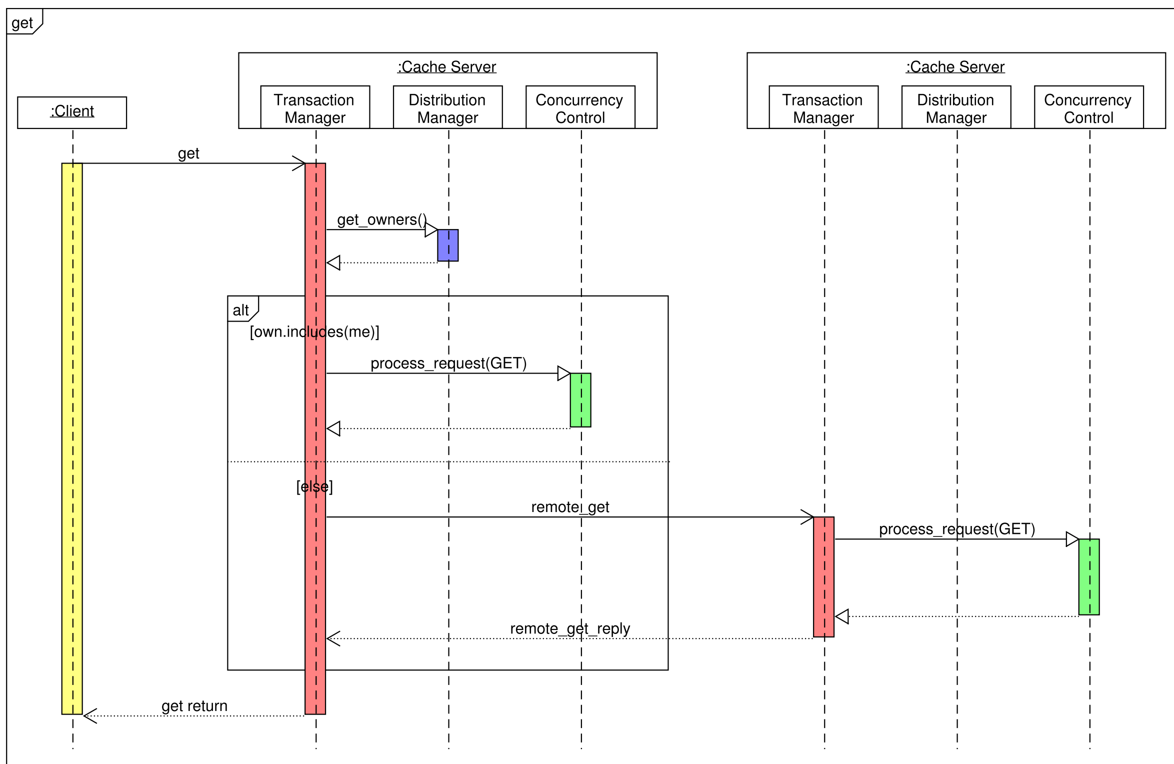

Fig. 3. Management of the get event.

implementing non-blocking read access to data (as typical of most data grid products guaranteeing weak data consistency, such as read committed or repeatable read semantics [1]), the read operation may anyway be blocked in case no local copy exists and needs to be fetched by some remote cache sever. This is automatically handled by our framework since the TM module records information on any pending simulated read operation within a proper data structure (this takes place in the 'else' part of the red-box in the diagram). When setting up the record for a given operation, information on the remotely-contacted cache servers, if any, is also installed. That record will be removed only after processing the corresponding reply simulation events from all those cache servers, which is done for allowing an optimized execution flow for those reply events. On the other hand, the operation is unblocked (and a reply event is scheduled towards the corresponding client) when the first copy of the data becomes available from whichever cache server, hence after processing the first remote_get_reply simulation event.

In Fig. 4 we show the sequence diagram associated with the management of put events (namely data-object updates). These events trigger the update of some meta-data locally hosted by the cache server (this takes place in the red-box in

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

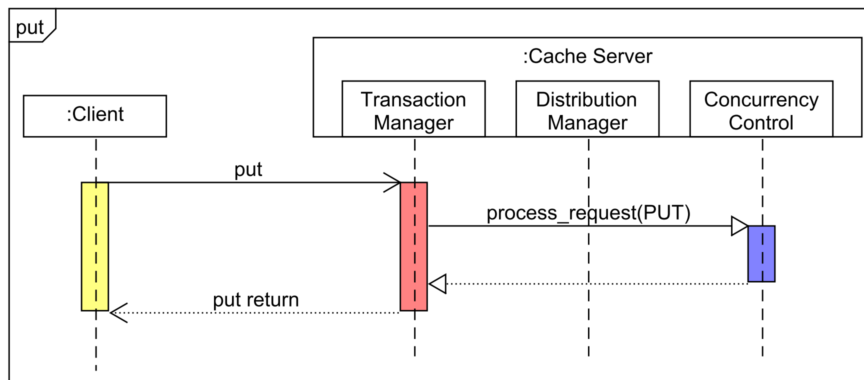

Fig. 4. Management of the put event.

the diagram), which keep the transaction write set into a record referred to as `TxWriteSet`. Such an update takes place after having invoked the CC module.

On the other hand, the meta-data are queried upon simulating a get operation to determine whether the data-object to be read already belongs to the transaction read/write set (hence whether the get operation can be served immediately via information within the read/write set). In such a case, the simulation-event pattern for handling the get is slightly different from the general one depicted above since it only entails simulating local CPU usage required for providing the data-object extracted from the transaction read/write set to the client. This implicitly leads the framework to provide support for simulating transactional data management protocols ensuring at least repeatable-read semantic.

3.1.3. The commit event

More complex treatments are actuated when handling commit simulation events incoming at the cache servers. Specifically, as shown by the sequence diagram in Fig. 5, the commit will result in scheduling prepare events towards all the cache servers that figure as owners of the data to be updated. Each of these events carries the keys associated with the data-objects to be updated, which are again retrieved via the `TxWriteSet` data structure maintained by the cache server acting as transaction coordinator. TM can determine the set of target cache servers by exploiting the keys associated with the written data-objects (which are kept within the transaction write set) by querying the DM module. In case the local cache server is one of the owners of the data, the interaction between the TM and the local CC takes place as a simple synchronous procedure call. In any case, the CC module exhibits the same simulated behavior independently of whether the prepare phase for the transaction needs to run local tasks on the same cache server, or remote tasks. Hence, the CC module operates seamless of any simulated data distribution/replication scheme. For the preparing transaction, the framework logs the identities of the contacted servers, and then waits for the occurrence of prepare_reply simulation events scheduled by any of these servers.

In case the prepare_reply events are positive from all the contacted servers, final commit events are scheduled for all of them, which will ultimately result in invocations of the CC module. On the other hand, abort events are scheduled in case of negative prepare outcome. Further, for the case of primary ownership, the commit events are propagated to the non-primary owners, in order to let them reflect data update operations.

Upon finalization of a transaction, TM automatically invokes the module `finalizeTransaction`, which receives as input the pointers to both `TxInfo` and `TxStatistics` records so as to allow for their update (particularly the statistics). The release of these buffers within the framework is again handled automatically. However, before releasing any of them, the module `statisticsLog` is called, passing as input pointers to both of them, allowing the modeler to finally log any provided statistical data.

3.1.4. Details on the CC module

Let us now detail the behavior of the CC simulation module. By the above description, this module is invoked by TM upon the occurrence of get or remote_get events, put events, prepare events, and commit events. The CC module is oblivious of whether a requested action is associated with some local or remotely-executed transaction. It takes the following input parameters:

- a pointer to the `TxInfo` record;
- a pointer to `TxStatistics` (or NULL if the cache server is not the transaction coordinator);
- the `type` of the operation to be performed (read, write, prepare or commit);
- the `key` of the data-object to be involved in the operation.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

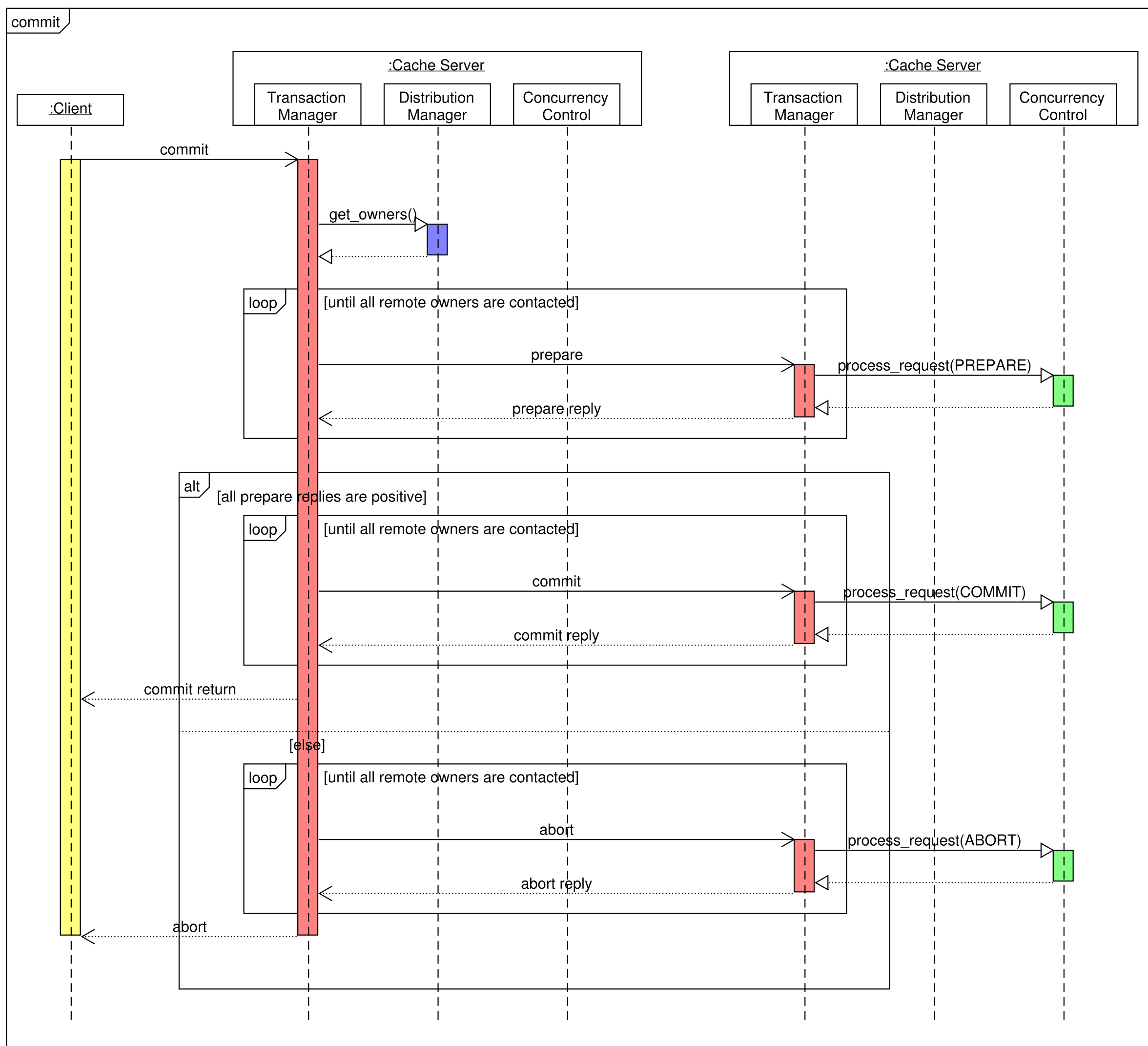

Fig. 5. Management of the commit event.

On the other hand, CC can reply to invocations from TM via the return values listed below:

- WAIT, indicating that the currently requested operation leads to a temporary block of the transaction execution;
- READ_DONE, indicating that the data-object can be returned to the reading transaction;
- WRITE_DONE, indicating that the write operation has been processed;
- PREPARE_DONE, indicating that the transaction has been successfully prepared;
- PREPARE_FAIL, indicating that the transaction prepare stage has not been completed correctly; and
- COMMIT_DONE, indicating that the transaction commit request has been processed.

Once the TM module takes back control upon the return of CC, the above return values trigger the generation of actual simulation events to be exchanged across different simulation objects. As an example, PREPARE_DONE and PREPARE_FAIL return values give rise to the scheduling of the aforementioned prepare_reply events, with proper payload (indicating positive or negative prepare outcomes).

Finally, a callback mechanism allows CC to notify to TM the change of the state of any previously blocked transaction (so as to allow TM to schedule, e.g, the prepare_reply event towards the transaction coordinator), and to request TM to schedule timeout events, which can be useful in scenarios where CC actions are also triggered on the basis of passage of time.

3.2. The client simulation-object

Client simulation objects have an internal structure that does not need to be changed by the simulation modeler. In fact, he only needs to specify, via configuration files within the framework, what type of probability distribution must be used for determining the data to be accessed, and what distributions need to be used for determining the number of operations to be executed within a transaction and the type (read or write) of each operation.

As for this aspect, the framework already offers the possibility to use differentiated access distributions, some of which are analytic, while others have been determined by relying on traces of known benchmarks. Further, the clients can be configured in order to simulate either an open or a closed system. For the former case, the simulation modeler needs to specify the rate of generation of transactions at the client side. As a final note, our client simulation object also embeds the possibility to generate the workload by directly relying on traces (rather than distributions derived from the traces).

3.3. Modeling message exchange dynamics via machine-learning

As hinted, our framework relies on black-box, machine-learning-based modeling techniques to forecast the dynamics at the level of the message-passing/networking sub-system. Developing white-box models (e.g. simulative models) capable of capturing accurately the effects by contention at the network level on message exchange latencies can in fact be very complex (or even non-feasible, especially in virtualized cloud infrastructures), given the difficulty to gain access to detailed information on the internals of messaging/network-level components [8].

As already mentioned, contention on the network layer, and the associated message delivery delay, can have a direct impact on the latency of two key transaction execution phases within the data grid, namely the distributed commit phase, and the fetch of data whose copies are not locally kept by the cache server, given that the whole data-set might be only partially replicated across the nodes (e.g. for scalability purposes). These latencies, in their turn, may affect the rate of message exchange, and so the actual load on the messaging system (in the simulated configuration of the workload and for the specific data grid settings).

More in general, estimating (hence predicting) the message transfer delay while simulating some data grid system deployed over a specific networking software/hardware (virtualized) stack boils down in our approach to a non-linear regression problem, in which we want to learn the value of continuous functions defined on multivariate domains. Given the nature of the problem, we decided to rely on the Cubist machine learning framework [27], which is a decision-tree regressor that approximates non-linear multivariate functions by means of piece-wise linear approximations. Analogously to classic decision-tree-based classifiers, such as C4.5 and ID3 [28], Cubist builds decision trees choosing the branching attribute such that the resulting split maximizes the normalized information gain. However, unlike C4.5 and ID3, which contain elements in a finite discrete domain (i.e., the predicted class) as leaves of the decision tree, Cubist places a multivariate linear model at each leaf.

Clearly, the reliance on machine-learning requires building an initial knowledge base in relation to the networking dynamics of the target virtualized infrastructure (as observable from the outside, in compliance with the black-box approach that characterizes machine-learning methods), for which we need to simulate the behavior of some specific data grid system (or configuration) run on top of it. This can be achieved by running (possibly once) a suite of (synthetic) benchmarks that generate heterogeneous workloads in terms of mean size of messages, memory footprint at each node, CPU utilization, and network load (e.g. number of transactions that activate the commit phase per second). As for this aspect, one could exploit some (open source) data grid system relying on the specific messaging layer for which the machine learner must provide the predictions. This approach looks perfectly suited for data-grid providers (namely for scenarios where the data-grid system is provided as a PaaS [29]), given that they can take advantage of (historical) profiling data related to specific (group) communication and messaging systems run on top of given (consolidated) virtualized platforms.

Also, it is well known that the selection of the features to be used by machine-learning toolkits plays a role of paramount importance, since it has a dramatic impact on the quality of the resulting prediction models. Such set of features has to be highly correlated to the parameters the machine learner is going to predict, namely the message transfer delay across nodes within the system. In the following, we list the set of features we selected, also motivating our choices:

- Used memory: it has been shown that the memory footprint of applications can affect significantly the performance of the messaging layer [5,8].
- CPU utilization: this parameter is required given that the message delivery latency predicted by our machine learner includes a portion related to CPU processing (such as the marshalling/unmarshalling of the message payload).
- The message size: this parameter is of course highly related to the time needed to transmit messages over the (virtualized) networking infrastructure.
- The number of message exchange requests per second: this parameter provides a good indicator of the network utilization.

Clearly, predicting metrics such as the message delivery latency under a specific simulation scenario depends on how the simulation model progresses, e.g., in terms of simulated system throughput and consequent actual number of message exchange operations per second (see the last parameter listed above). These parameters, as well as others (like the average

size of exchanged messages), are in their turn targeted in the estimation process by simulation. Hence they might be unknown at the time the machine learner is queried during the simulation run.

This problem is intrinsically solved by the specific way we couple simulative and machine learning components. Particularly, when a prediction on the delay of message delivery is required for a specific message send operation, the simulative components compute (estimate) the values needed as input by the machine learning component, depending on the current simulated system state. This is done easily and efficiently given that in our framework all the values of the parameters required in input by the machine learner (e.g. the current CPU utilization) to carry out its prediction are constantly updated, hence they are readily available. By using these values, the actual query to the machine learner is issued to determine the timestamp of the discrete-event associated with the message delivery along the simulation time axis. This coupling scheme is depicted in Fig. 6, and the actual implementation of this kind of interaction within our framework has been based on linking Cubist as a library directly accessible (invocable) by the simulation software.

This coupling approach leads the machine learner to output "updated" predictions for the message transfer delay (as a function of the message size), while the simulation run approaches the steady state value of the target parameters to be estimated (e.g. the system throughput, which may in turn depend on parameters like CPU usage). Hence, the process of "rejuvenating" the predictions by the machine learner ends upon converging towards the actual final estimation of the target parameters by the simulation run.

3.4. A final overview of the framework architecture

In Fig. 7 we present the component diagram of our proposed framework. As mentioned, the essential building blocks are the *Client* and the *Cache Server* components. The former relies on the *Configuration* & *Knowledge Base Manager* component, which is in charge of managing the configuration file (used by the framework to initialize the simulation and determine how data access patterns should be driven). Depending on the actual configuration, the *Client* component interacts with either the *Trace* or the *Generator* component. The former is in charge of telling the *Client* what is the actual data access pattern depending on a real-word trace file which *Trace* is able to parse and manage. The latter, on the other hand, randomly generates different data access patterns depending on the configuration of the framework.

The *Client* component interacts with the *Discrete Event Simulation Engine* by means of Process and Schedule ports. While our implementation of the *Discrete Event Simulation Engine* relies on an optimized sequential calendar-queue-based scheduler, this component can be easily replaced by any available Discrete-Event System, provided that the used ports (and their interface-specification) are the same. The *Discrete Event Simulation Engine* component lets the *Client* and *Cache Server* components exchange messages and process them in a timestamp-ordered way, thus ensuring consistency of the overall simulation. The *Cache Server* component, on its turn, implements all the already described facilities, and explicitly interacts with the *Cubist* component so as to retrieve Message Latency Predictions, associated with, e.g., network delays. In order for *Cubist* to provide predictions which are representative of the currently-simulated system, it is connected to the *Configuration* & *Knowledge Base Manager* component to retrieve the knowledge base related to the current deployed system, which is in turn used by *Cubist* to make actual predictions. The *Cache Server* component also interacts with the *Configuration* & *Knowledge Base Manager* component in order to get information on the selected policy for placing data-object copies across the different cache servers. This is required in order to determine data ownership. In our current implementation *Configuration* & *Knowledge Base Manager* already offers the support for classical placement policies such as consistent hashing based ones [37].

By the framework structure, the only component that needs to be modified in order to build models of different data grid systems (e.g. based on different distributed coordination schemes) is the *Cache Server* component. Also, by the capabilities that are already offered by the actual architecture of this component, re-modeling can be done by only dealing with transaction identifiers, basic transaction setup information and relations across different transactions, on the basis of the actual data-objects locally hosted by a given cache server. This is a relevant achievement when considering that great research effort is currently being spent in the design of concurrency control algorithms suited for cloud data stores, which provide differentiated consistency vs scalability tradeoffs (see, e.g., [22–26]), each one fitting the needs of different application contexts. Having the possibility to provide simulation models of such differentiated algorithms by exploiting our framework can definitely reduce the time and effort required for assessing their potential.

To determine what are the locally hosted data-objects, hence the locally hosted keys, the CC module within the *Cache Server* component accesses a hash table that gets automatically setup upon simulation startup (as hinted, by exploiting the Data Placement port of the *Configuration* & *Knowledge Base Manager* component). On the other hand, the meta-data required to keep relations across active transactions, (e.g. wait-for relations), and the corresponding data structure is completely left to the simulation-modeler. However, the actual instance of this data structure can be accessed via a special pointer which is passed to the CC module by the framework as an additional input parameter. We note that if the pointer value is `NULL`, then CC has not yet allocated and initialized the structure, hence this must be done, and the actual pointer to be used in subsequent calls to CC can be setup and returned upon completion of the current CC execution.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

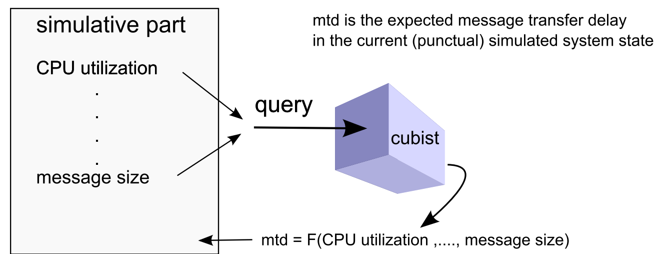

Fig. 6. Coupling of simulative and machine learning components.

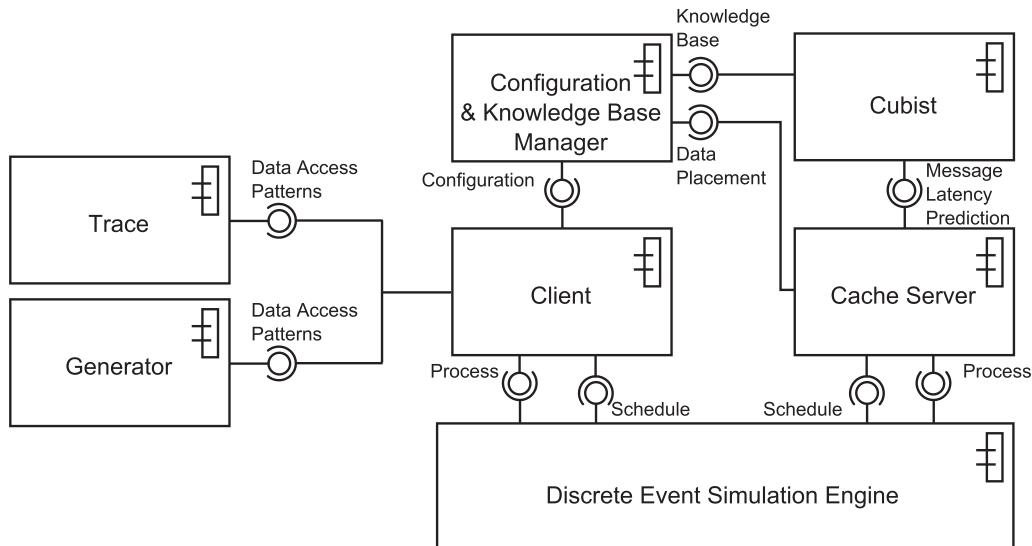

Fig. 7. Components diagram.

4. Experimental study: framework validation and usage

The skeleton operations described in the former section, such as the ones related to 2PC coordination, compose the foundational/base simulative model of our framework, which users can extend and customize to meet their needs. For this reason, we have decided to provide validation data for a case study where we compare the framework outputs against real data achieved by running a data grid system exactly exploiting such an archetypal 2PC coordination paradigm. In particular, we present data from a study where we compare simulated performance results with the corresponding ones achieved by running the 2PC-based Infinispan data grid system by JBoss/Red-Hat [1].

Our experimentation has been based on a wide spectrum of system settings given that we consider large scale deployments on top of both public and private cloud systems. Further, the workloads generated in our tests are based on various configurations of the YCSB benchmark by Yahoo [10]. Given that this benchmark has been devised just to assess (cloud suited) in-memory data stores, its employment further contributes to the relevance of the selected experimental configurations.

The last part of this section is devoted to presenting the results of an experimentation where the framework is exploited for on-line capacity planning and reconfiguration purposes. The outcome by this part of the experimental study complements the validation data we report.

4.1. The Infinispan data grid and its integration with the framework

Infinispan is a popular open source in-memory data grid developed in Java currently representing both the reference data platform and the clustering technology for JBoss, which is the mainstream open source J2EE application server. Infinispan exposes a pure key-value data model, and maintains data entirely in main-memory relying on replication as its primary mechanism to ensure fault-tolerance and data durability. As other recent (NoSQL) platforms, Infinispan opts for weakening consistency in order to maximize performance. Specifically, it does not ensure serializability [30], but only guarantees the Repeatable Read ANSI/ISO isolation level [31]. At the same time, atomicity of distributed updates is achieved via 2PC. This is used to lock all the data-objects belonging to the write-set of the committing transaction, so as to atomically install

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

the corresponding new data versions. The old committed version of any data-object remains anyhow available for read operations until it gets superseded by the new one.

In the Infinispan configuration selected for our experiments, the 2PC protocol operates according to a primary-owner scheme. Hence, during the prepare phase, lock acquisition is attempted at all the primary-owner cache servers keeping copies of the data-objects to be updated. If the lock acquisition phase is successful, the transaction originator broadcasts a commit message, in order to apply the modifications on these remote cache servers, which are propagated to the non-primary owners.

Clearly, the integration of our simulative framework with a real product such as Infinispan, for either validation or operations like (on-line) capacity planning of the data grid system, requires specific steps. In what follows we presents the steps we have carried out, which led the framework to be fully integrated within the Cloud-TM open source platform [29], namely a data grid platform entailing (on-line) reconfiguration and optimization capabilities. Clearly, our explanation can be used as a reference for the integration and usage of the framework in systems based on data grid products other than Infinispan.

4.1.1. Infinispan core internals

Infinispan relies on an architecture based on *Commands* and *Managers*. A Command is a Java object that represents a single operation, e.g., a commit transaction; a Manager is a software component that is responsible for carrying out operations needed to execute Commands, e.g., acquiring locks. Infinispan implements the *visitor* pattern [32] to relate Commands and the corresponding actions to be executed by Managers. In particular, an *Interceptor* is a Java object that serves the role of *visitor*, i.e., it acts as a hook between a Command *cmd* and the set of Managers that have to perform some operations to bring *cmd* to completion. A *visitCmdCommand* method is exposed by an Interceptor to register the logic that is in charge of dealing with a command of type *cmd*, e.g., CommitCommand.

For example, the LockingInterceptor deals with everything pertains locks, e.g., interacting with a LockManager for their acquisition and release, whereas the RPCManager handles the interactions with remote nodes, e.g., by invoking remote procedure calls corresponding to remote gets and dissemination of prepare/commit/rollback operations. Simplified snippets of code illustrating the tasks performed by the LockingInterceptor and the LockManager are provided, respectively, in Figs. 10 and 8.

In a dual fashion, the CommitCommand is visited by the LockingInterceptor to trigger the release of locks upon the successful execution of a transaction and by the DistributionInterceptor to communicate the successful outcome of the transaction to other involved nodes. Fig. 11 provides a simplified example of a Command implementation, by reporting how the CommitCommand accepts visiting Interceptors.

Interceptors are layered in a *chain*: when a transaction issues an operation, the corresponding Command is created and pushed down the interceptors chain until completion, i.e., until it has been visited by every interceptor for which it has registered a corresponding hook. This scheme also applies to operations coming from remote nodes: for example, upon receiving the request to serve a remote get, the corresponding command is created on the receiving node and pushed down the Interceptors chain to be executed.

4.1.2. Collecting run-time data in Infinispan

Infinispan natively keeps track of basic performance indicators, like the time it takes to complete a two-phase commit or the ratio between local and remote operations. On top of that, we have extended Infinispan's architecture so as to collect a number of additional low-level statistics aimed at monitoring the behavior of the running application over time and at characterizing its workload (see Fig. 9). This is needed in order to apply the proposed framework not only for performing off-line what-if analysis and capacity planning (for a real system), but also for carrying out automatic optimization and resource provisioning at runtime. The collection of statistics is performed transparently to the running application by injecting lightweight software probes: these are aimed at measuring CPU demands and response time of various operations, e.g., local/remote gets, prepare phases, lock hold times, and at collecting data access pattern information, e.g., data item popularity, number of puts/gets performed by transactions. These probes are inserted either at the Interceptor level, by means of specific Interceptors, or at the Manager level, by means of dedicated wrappers, as sketched in Figs. 8 and 12.[5] Aggregated statistics collected on an Infinispan node are stored in an ad-hoc data-structure, which can be queried by external processes via the JMX (Java Monitoring Extensions) technology.[6] The gathered information can be easily accessed either for visual inspection and monitoring, for example with simple tools like JStat,[7] or acquired to be further processed (e.g. averaged) so as to be given as input to the proposed simulation framework.

To this end, we have implemented a distributed monitoring system based on the Lattice Monitoring Framework, that has been widely used in successful cloud monitoring/management projects such as the well known RESERVOIR project [34]. This has been extended in order to gather not only statistics relevant to the utilization of physical resources, e.g., CPU and RAM, but also to collect the set of measurements taken at the Infinispan level, which (as pointed out) are exposed via JMX. On a periodical basis, statistics concerning load, resource utilization and workload characterization across the set of nodes in the data grid are collected by the distributed monitoring system and conveyed to an aggregator module. This module is

[5] Further details about the collected statistics may be found in [33].
[6] http://www.oracle.com/technetwork/articles/java/javamanagement-140525.html.
[7] http://docs.oracle.com/javase/7/docs/technotes/tools/share/jstat.html.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

```
public class LockManagerWrapper implements LockManager {
        private LockManager actualLockManager; //Actual LockManager implementation

        public LockManagerWrapper(LockManager actual) {
      this.actualLockManager = actual;
   }

   public boolean acquireLock(Object key,...)  {
          boolean locked;
          //Obtain the statistics' container for the current transaction
      TransactionStatistics transactionStatistics =
                    TransactionsStatisticsRegistry.getCurrentTransactionStatistics();
      locked = actual.acquireLock(key, ...);  //invoke the actual LockManager
      if (locked) {
         transactionStatistics.addTakenLock(key); //Keeps track of the acquired locks
      }
      else{
         updateContentionStats(transactionStatistics, key); //Track lock contentions
      }
      return locked;
   }
   ...
}
```

Fig. 8. The LockManagerWrapper collects locks-related statistics; it encapsulate the original Infinispan implementation of the LockManager, so as to avoid changing its internals.

```
class TransactionStatistics{
   long initTime; //Wall-clock timestamp corresponding to the begin of the transaction
   long endLocalTime; //Wall-clock Timestamp set upon the distributed commit phase
   long initCpuTime; //CPU Timestamp corresponding to the begin of the transaction
   long endLocalCpuTime; //CPU Timestamp set upon reaching the distributed commit phase
   Map<Object, Long> acquiredLocks = new HashMap<Object, Long>(); //Track locks' hold-time
   ...
}
```

Fig. 9. The TransactionStatistics maintains low-level statistics about a single transaction. It is retrieved by invoking the static method getCurrentTransactionStatistics() of the class TransactionsStatisticsRegistry (not shown), so as to guarantee it is accessible from any point in the code.

```
class LockingInterceptor extends AbstractLockingInterceptor implements Visitor{
   private LockManager lockMager;
   public Object visitCommitCommand(CommitCommand command,TxInvocationContext ctx)
           throws Throwable {
      try {
      return super.visitCommitCommand(command,ctx); //Call parent's method
      } finally {
         lockManager.unlockAll(ctx); //Release all locks
      }
   ...
}
```

Fig. 10. The LockingInterceptor visits Command objects to specific lock-related operations of the LockManager.

```
class CommitCommand{
   public Object acceptVisitor(InvocationContext ctx, Visitor visitor) throws Throwable {
      return visitor.visitCommitCommand((TxInvocationContext) ctx, this);
   }
   ...
}
```

Fig. 11. The CommitCommand represents the commit operation of a transaction. It accepts Visitors, like the LockManagerInterceptor.

responsible for aggregating and processing statistics, e.g., by computing average values, and for encoding them in a format that is consumable by the simulation framework to answer what-if queries. The overview of the final architecture is presented in Fig. 13, which shows how a set of (virtual) machines and the hosted instances of Infinispan cache servers

ARTICLE IN PRESS


```
@MBean(objectName = "Statistics",
       description = "Component managing extended statistics relevant to transactions.")
class StatsInterceptor{
  public Object visitCommitCommand(TxInvocationContext ctx, CommitCommand command)
           throws Throwable {
   //Obtain the statistics' container for the current transaction
      TransactionStatistics transactionStatistics =
                TransactionsStatisticsRegistry.getCurrentTransactionStatistics();
      transactionStatistics.setTransactionOutcome(true); //Set the transaction as completed
      ...
      Object ret = invokeNextInterceptor(ctx, command);  //Invoke following interceptors
      TransactionsStatisticsRegistry.terminateTransaction(transactionStatistics);
                                                           //Gather statistics
      return ret;
}

@ManagedAttribute(description = "Average number of puts by a successful transaction",
           displayName = "No. of puts per successful local transaction")
   public double getAvgNumPutsBySuccessfulLocalTx() {
      //Retrieve the required statistic and return it
   }
   ...
}
```

Fig. 12. The StatsInterceptor is a custom Interceptor, designed by us to expose the gathered statistics via JMX and to collect some information by visiting commands. Annotations at the class and the method level are used by Infinispan to enable interaction with the class objects via JMX.

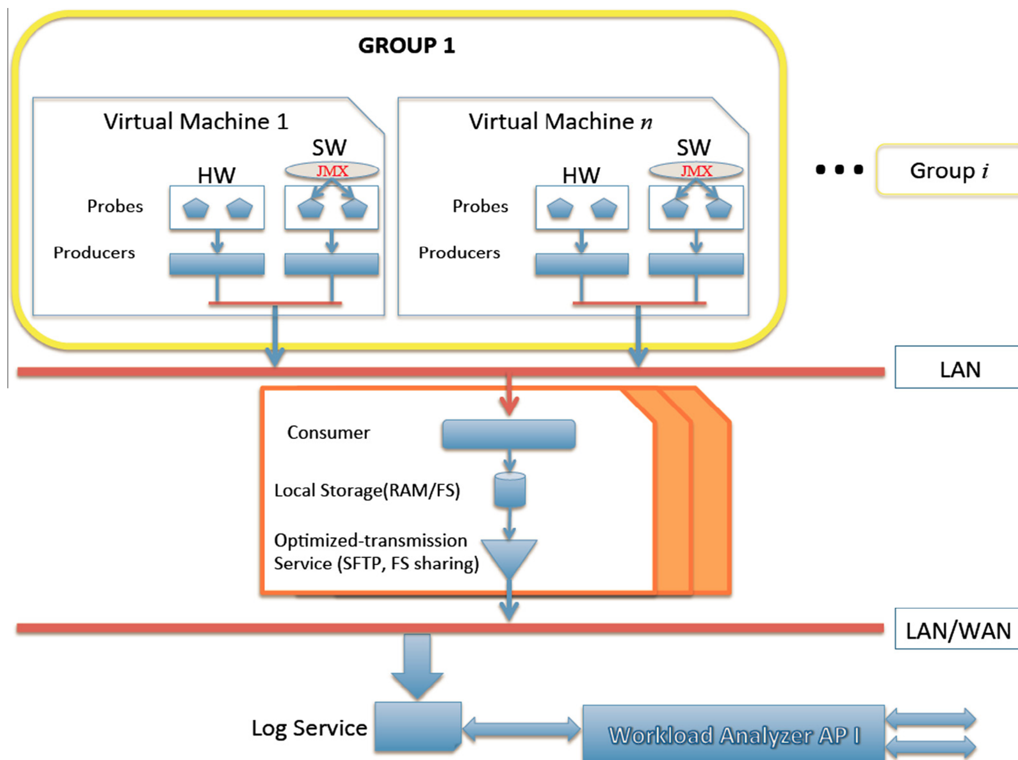

Fig. 13. Architectural organization for (run-time) data acquisition and analysis.

(the software – SW – part in the picture) are monitored by using multiple Lattice producers and a single Lattice consumer. Also, the raw-data that are gathered via the log-service are then filled in input to a workload analyzer subsystem (acting as the aggregator) that is in charge of computing statistically representative average values, and is also in charge of running time-series analysis so as to embed capabilities of predicting variations of, e.g., the workload intensity. The latter aspect can be particularly interesting for on-line capacity planning, where prediction models as the ones offered by our framework can be queried by filling them with the predicted workload in order to determine well suited configurations to be put in place just to cope with expected workload variations.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

4.2. Cloud infrastructures exploited in the validation study

The experimental test-bed for our validation study consists of a private and a public cloud infrastructure. The Virtual Machines (VMs) deployed on both clouds are equipped with 1 Virtual CPU (VCPU) and 2 GB of RAM. They all run a Fedora 17 Linux distribution with kernel 3.3.4.

The private cloud consists of 140 VMs deployed over a cluster composed of 18 machines equipped with two 2.13 GHz Quad-Core Intel(R) Xeon(R) processors and 32 GB of RAM and interconnected via a private Gigabit Ethernet. Openstack Folsom is employed to regulate the provisioning of resources and Xen is used as virtualization software. The public cloud consists of 100 VMs, deployed over the FutureGrid India infrastructure [35], which exploits the Openstack Havana virtualization software.

4.3. Workload configurations

We have implemented an extended version of YCSB where transactions involving multiple individual get/put operations, as natively specified by the benchmark, are supported. We rely on three different workload configurations, which we refer to as A, B and F. Workload A has a mix of 50% read and 50% write (namely update) transactions; workload B contains a mix with 90% read and 10% write transactions, while in workload F records are first read and then modified within a transaction. Also, we have carried out experiments with two different data access profiles. In the first case, the popularity of data items follows a zipfian distribution with YCSB's zipfian constant set to the value 0.7. In the second one, which we name hot spot case, 99% of the data requests are issued against the 1% of the whole data set. A total amount of 100,000 data-objects constitutes the data set in all the experiments.

In the plots, we will refer to a specific workload configuration using the notation N–D–P–I, where: 'N' refers to the original workload's YCSB notation [10]; 'D' is the number of distinct data items that are read by a read-only transaction; for update transactions, it is the number of distinct data items that are written (for the 'F' workload, which exhibits a read-modify pattern of update transactions, any accessed data is both read and written); 'P' encodes the data access pattern ('Z' stands for zipfian, 'H' for hot-spot); finally, 'I' specifies the cloud infrastructure over which the benchmark has been run ('PC' stands for private cloud, 'FG' for FutureGrid).

4.4. Validation results

All the above illustrated workload configurations have been run on top of the selected cloud systems while scaling the number of VMs, and relying on a classical consistent hashing [37] scheme for placing the data copies across the servers. The run outcomes have been exploited both to collect statistically relevant values for core performance parameters in the real system and to determine the value of the input parameters for the simulated data grid (e.g. the CPU demand for specific operations, to be eployed at the level of the simulation models).

As a final preliminary note, in the real system the workload generator has been deployed as a thread running on each VM, which injects requests against the co-located Infinispan cache sever instance, in closed loop. Consequently, in the simulation model configuration, no networking/messaging delays have been modeled between clients and cache server instances. Yet, the (simulated) networking/messaging system plays a core role in the data exchange and coordination across the different cache server instances. This well fits the relevant scenarios where the focus of performance analysis/prediction is on sever side infrastructures.

The validation has been based on measuring the following set of Key Performance Indicators (KPIs), and comparing them with the ones predicted via simulation: (i) the system *throughput*, (ii) the transaction *commit probability* (this parameter plays a role for update transactions, given that read-only transactions are never aborted by the concurrency control algorithm considered in this study), and (iii) the *execution latency* of both read-only and update transactions. The first KPI in the above list provides indications on the overall behavior of the system, and hence on the accuracy of the corresponding prediction by the framework-supported models. The second one is more focused on the internal dynamics of the data grid system (e.g. in terms of the effects of the distributed concurrency control mechanism), which have anyhow a clear effect on the final delivered performance. Finally, the execution latency of the different types of transactions has been included in order to provide indications on how the simulative models are able to reliably capture the dynamics of different kinds of tasks (exhibiting different execution patters) within the system. In fact, read-only transactions can require remote data fetches across the cache severs but, differently from update transactions, they entail no 2PC step. Simulation runs have been stopped as soon as collected statistics in subsequent periods were tracked to provide 95% confidence interval on the core estimated metric, namely the system throughput. Also, the running times of simulations on top of an off-the-shelf Core-i7 processor equipped with 8 GB RAM, running Linux (kernel 2.6), were very limited. In fact, simulations terminated in less than 10 s for instances of the problem up to the order of ten cache servers, while a bit more time has been requested by larger scenarios. Anyhow, simulation execution times were bounded by 1 min for most of the more complex configurations.

The results for the case of data grid deploy on top of the private cloud system are reported in Fig. 14. For all these tests we considered a configuration where each data-object is replicated two times across the cache servers, which is a typical settings allowing for system scalability, especially in contexts where genuine distributed replication protocols are used to

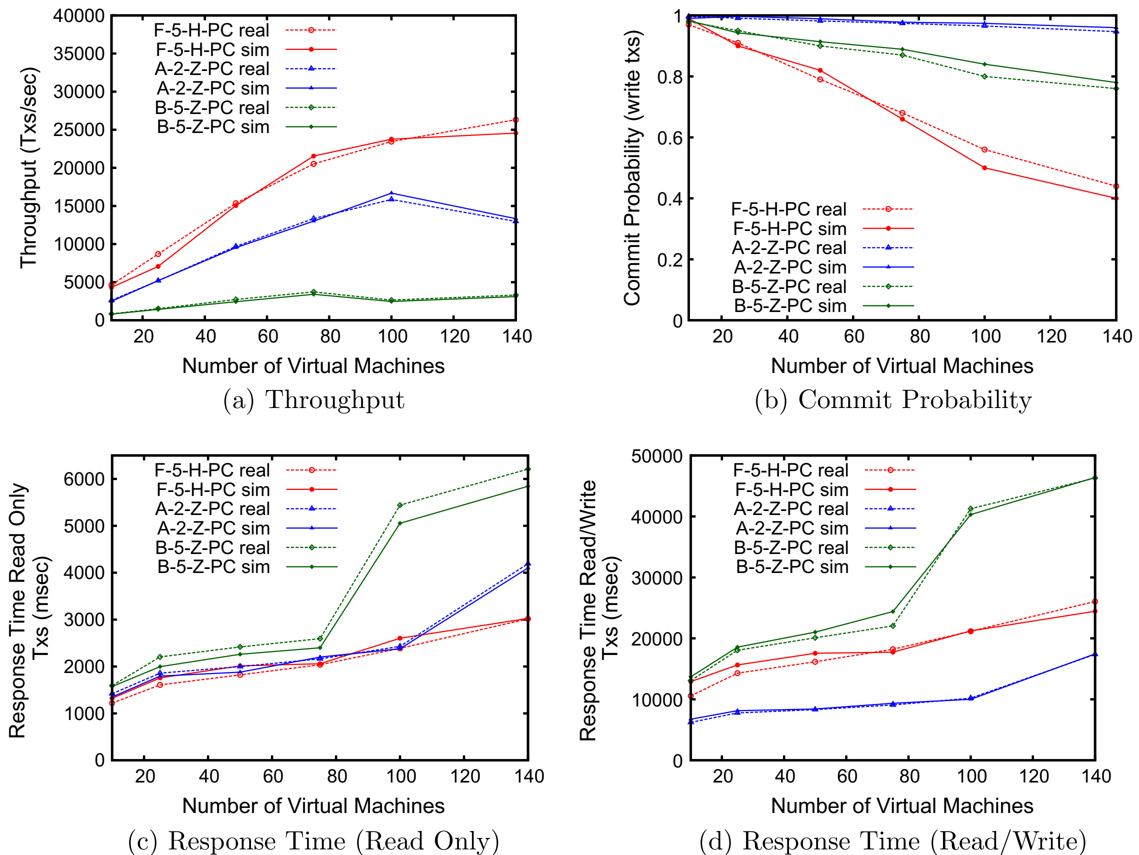

Fig. 14. Results for deploy on the private cloud (up to 140 VMs).

manage the data access [38,26].[8] Therefore, this value well matches the nature of this particular validation study, given that we consider deployments on large scale infrastructures (up to 140 VMs). Also, the variance of simulation results across different runs (executed with different random seeds) is not explicitly plotted given that the obtained simulative values were quite stable, differing by at most 10%.

By the plotted curves we can see how the KPIs' values predicted via simulation have a very good match with the corresponding ones measured in the real system, at any system scale. As an example, the maximum error on the overall throughput prediction is bounded by 20%, as observed for the configuration F-5-H-PC when running on top of 25 VMs. However, except for such a peak value, the error in the final throughput prediction is for most of the cases lower than 5%. Similar considerations can be drawn for the other reported KPIs.

Another interesting point is related to the fact that the simulative models are able to correctly capture the real system dynamics when changing the workload. As an example, while we observe higher commit probability for an individual run of an update transaction in the scenario with the 50%/50% read/write mix and zipfian data accesses, the hot spot configuration allows for higher throughput values even though the update transaction commit probability is lower. This is clearly due to the fact that in the used hot spot configuration only 5% of the whole workload consists of update transactions that, although being subject to retries due to aborts with non-minimal likelihood (especially at larger system scales), impact the system throughput in a relatively reduced manner.

The results achieved for the case of deploys on top of the FutureGrid public cloud system, which are reported in Fig. 15, additionally confirm the accuracy of the models developed via the framework. In these experiments we further enlarge the spectrum of tested scenarios not only because we move to a public cloud, but also because (compared to the case of private cloud deploy) we consider a different value for the replication degree of data-objects across the servers, namely 3. This value leads to the scenario where fault resiliency is improved over the classical case of replication on only 2 cache servers, which is the usual configuration that has been considered in the previous experiments. By the data we again observe very good match between real and simulative results. Further, similarly to the previously tested configurations, such a matching is maintained

[8] A distributed transactional replication protocol is said to be genuine if it requires contacting only the nodes handling the data copies accessed by a transaction in order to manage any phase of the transaction execution, including its commit phase.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

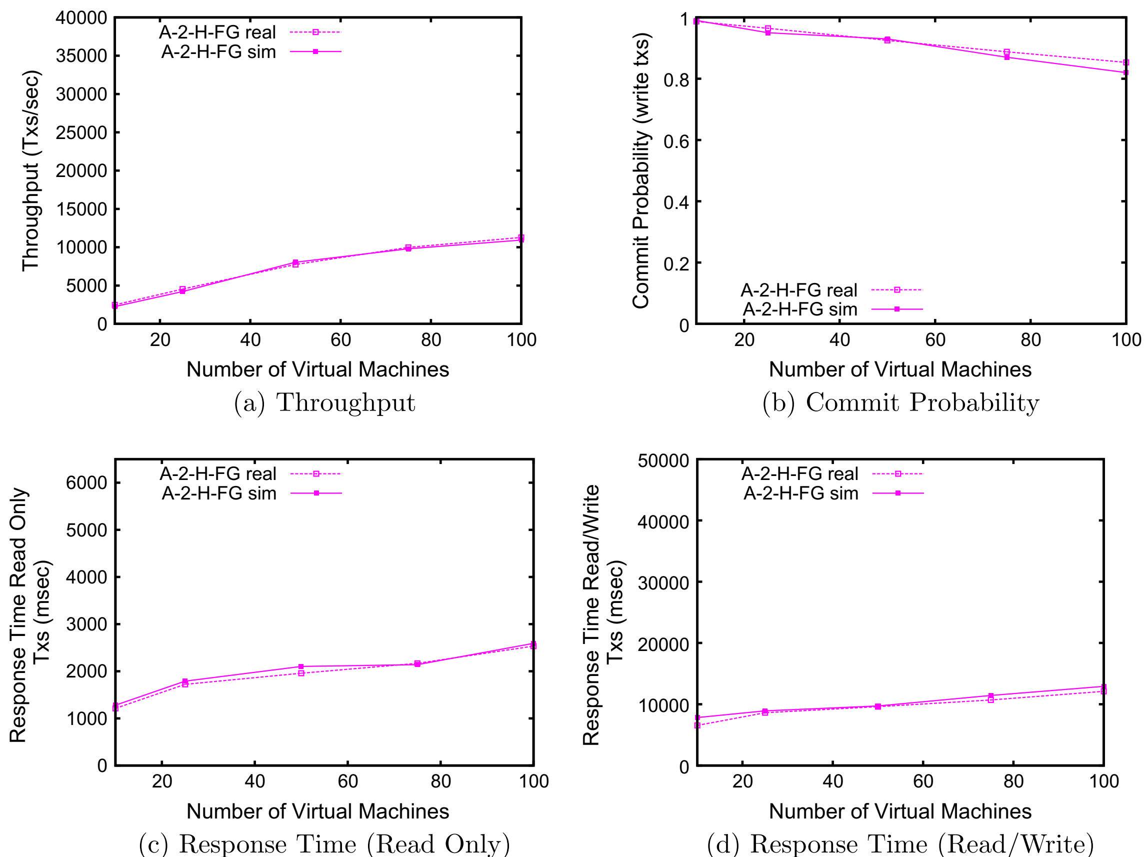

Fig. 15. Results for deploy on FutureGrid (up to 100 VMs).

at any system scale, and, importantly, also when the actual dynamics of the data grid system significantly change while scaling the system size. In fact, we observe that the commit probability of update transactions significantly changes when increasing the system scale. This phenomenon, and its effects on the delivered performance, are faithfully captured by the simulator. This is a relevant achievement when considering that the workload used for the experiments on top of the FutureGrid public cloud system has been based on a 50%/50% read/write transactions' mix, which leads transaction retries to play a relevant role on the final performance given that half of the workload can be subject to abort events, which become increasingly frequent at larger scales of the system.

As a final part of our validation study, we assess the effectiveness of the Machine Learning (ML) based approach to build models to forecast the latency of network-bound operations. To this end, we evaluate the accuracy achieved by the Cubist model when trained over the data-set corresponding to the message delivery latency across instances of the Infinispan cache serves deployed over both our private cloud infrastructure and FutureGrid.

The data-sets corresponding to both the private and the public cloud deployments are composed by around 300 samples each, obtained by deploying the Infinispan data grid platform over a different number of nodes, configuring it to use different replication degrees and while varying the generated workload (e.g. in terms of written data items, size of the messages and duration of the transactions' business logic).

Note that the workloads employed to build the knowledge base for the Cubist regressor do not entail contention of data. In fact, the proposed framework relies on a detailed simulation of the concurrency and replication control scheme to predict the impact of data contention on performance. This allows us to build an accurate performance predictor of the message delivery latency by requiring a very limited number of training data for the construction of the embedded ML model, compared to the amount that a pure ML-based predictor would need to learn the overall performance function, which strongly depends also on data contention [5,15].

The accuracy of the Cubist model built over a data-set S is measured by means of a 10-fold cross-validation. This entails partitioning S into 10 bins $S_1 \dots S_{10}$ and then, iteratively for $i = 1 \dots 10$, training Cubist over $S \setminus S_i$ and evaluating its accuracy against S_i. Fig. 16 reports the scatter-plots obtained when comparing the real and the predicted values for message delivery latencies, obtained by means of such 10-fold cross-validation; in particular, Fig. 16(a) refers to the private cloud Infinispan deployment and Fig. 16(b) to the FutureGrid one.

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

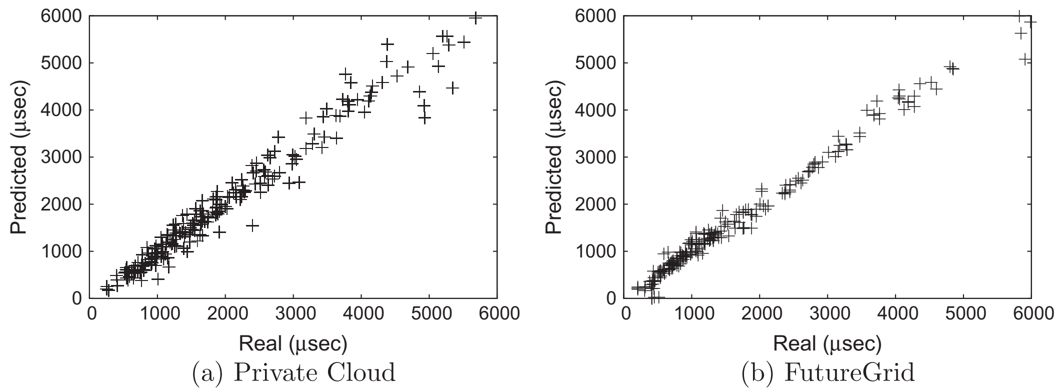

Fig. 16. Scatter plot evaluating the 10-fold cross-validation accuracy of the Cubist models that predict the message delivery latency.

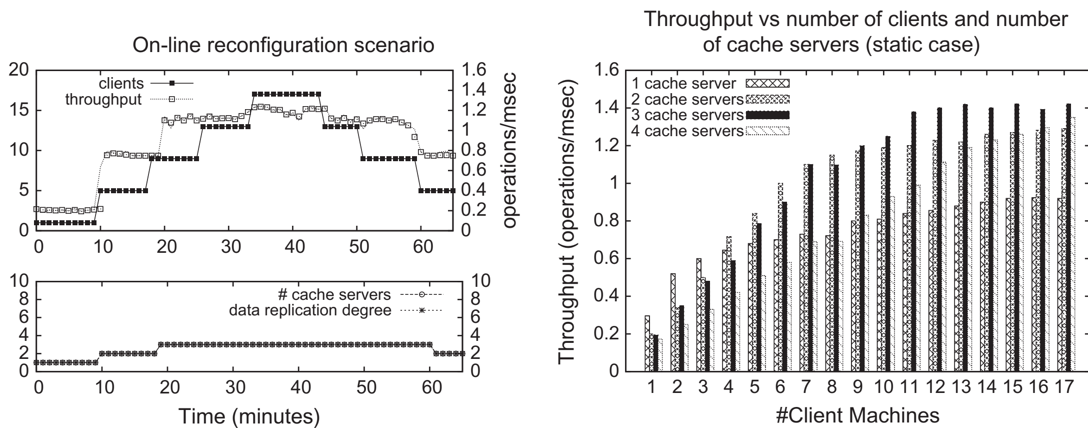

Fig. 17. Results for the on-line reconfiguration scenario.

The plots show that there is a very good agreement between the predicted and the observed values, demonstrating the capability of Cubist to infer an accurate performance model for the target metric. In order to quantitatively assess accuracy, we provide information about the correlation coefficient and the relative error computed by Cubist when evaluating the models built during the 10-fold cross-validation. The correlation coefficient is a metric that quantifies the degree of agreement between the real and the predicted values [27]. The relative error, instead, is computed as the ratio of the average error magnitude to the error magnitude that would result from always predicting the mean value [27]. We report that, for the considered data sets, the correlation coefficient achieved by Cubist is 1 in both cases, which means perfect agreement between the actual measurement and the values obtained by querying the Cubist models. This agreement can be witnessed by looking at Fig. 16, where the points in both the plots, corresponding to predicted values, are tightly cluttered around a fictitious $y = x$ line, which corresponds to perfect predictions. The accuracy of the Cubist models is also confirmed by very low values for the relative error metric, which we report to be only 0.05 for both the considered infrastructures.

4.5. On-line capacity planning

In this final part of our experimental study we provide data related to the usage of the framework for on-line capacity planning and reconfiguration purposes. We still rely on a variation of the aforementioned YCSB benchmark, particularly on a setting resembling the one used in [15] based on 10,000 data-objects, where each put operation modifies a single data-object, while each read-access can read a list of data-objects (hence it can be a query on a range of key values). The size of this list varies, with uniform probability, between 1 and 40 objects. The accesses of the client to the data-objects are uniformly distributed, with 2% of the accesses occurring in write mode, which gives rise to a limited contention scenario, as typical of several real life contexts.

For this part of the experimentation we relied on cloud resources offered by Amazon [36], thus further enlarging the spectrum of infrastructures used in our study. Particularly, clients and Infinispan cache servers were deployed onto *small EC2 instances* equipped with 1.7 GB of RAM and one virtual core providing the equivalent CPU capacity of 1.0–1.2 GHz 2007

Opteron or 2007 Xeon processors. Each machine runs Linux Ubuntu 10.04 with kernel 2.6.32-316-ec2. The standard Amazon load balancer has been used to dispatch to the server nodes the load generated by the client threads. Also, each client actually mimics the behavior of a front-end server, which accesses the in-memory data layer hosted by the back-end cache servers. In fact, each client continuously executes accesses to the data layer with no think time between subsequent operations, hence giving rise to a sustained workload, as if it was concurrently handling multiple interactions by end-clients, characterized by non-zero think time.

The on-line capacity planning scenario we consider is based on imposing a constraint on the cost of the infrastructure, hence on the maximum number of VMs used for hosting the data grid cache servers, while targeting the maximization of the system throughput. This number has been set to 4. Also, for fault tolerance purposes, we set the minimal replication degree of data to the value 2. The on-line reconfiguration of the system over time (namely, of the actual number of cache servers and replication degree – while still matching the constraints) is actuated on the basis of the outcome by the framework. For this experiment we consider an already established knowledge base in terms of network dynamics, to be exploited by the machine-learning part of the prediction models, while we resort on on-line monitoring of the workload parameters to be filled in input to the simulative part of the framework. Also, on-the-fly what-if analysis and reconfiguration is carried out each time the workload (depending on the number of client machines injecting requests towards the cache servers, which we have varied over time) exhibits a variation leading it to fall out of the 10% with respect to its value along the last observed window (where a window is set to be of 1 min in length).

The results for this test are shown in Fig. 17, where we report how the on-line reconfiguration scheme based on the framework output adjusts the actual number of cache servers to be used, and the data replication degree (graph at the bottom left of the figure), in face of variations of the number of clients (graph at the top left of the figure, left vertical axis). In the same graph, we also show the system throughput (right vertical axis). In order to validate the framework based on-line reconfiguration choices, in the same figure (right side), we report the throughput we measured using static configurations of the system, entailing a fixed number of clients and cache servers (the replication degree is not explicitly plotted given that, in all our tests, it always matches, in the best case, the corresponding number of servers, which is exactly the case for the replication degree selected by the on-line choices based on the framework what-if analysis). By the data, we get that the framework always allows to predict and put in place configurations (while varying the number of clients) which are aligned with the corresponding optimal static configurations when considering whichever fixed amount of client machines in the interval between 1 and 17. Also, the identification of the optimal configuration takes place very promptly (thanks to the very reduced time required by the simulations), which leads to minimal delay for achieving optimal throughput (e.g. by adding new instances of cache servers) when a ramp up of the workload is experienced, and to a minimal delay for the shrink of the number of cache servers to the optimal value when a ramp down of the workload occurs.

5. Conclusions

In this article we have presented a simulation framework for predicting the performance of (cloud) in-memory data grid systems which can be used for, e.g., what-if analysis aimed at the identification of the configurations (such as the number of Virtual Machines to be employed for hosting the data grid system under a specific workload profile) matching specific cost-vs-benefit tradeoffs. The design of the discrete event simulative framework has been based on the use of flexible skeleton models, which can be easily extended/specialized to capture the dynamics of data grid systems supporting, e.g., different distributed coordination schemes across the cache servers in order to guarantee specific levels of consistency for transactional data manipulation. The adequacy of the framework, and of its model instances, in predicting the dynamics of data grid systems hosted in cloud environments is a result of the combination of the discrete event simulative approach with machine learning. In our framework architecture, the latter modeling technique is used to predict the dynamics at the level of networking/messaging sub-subsystems which, in cloud contexts, are typically unknown in terms of their internal structure and functioning, and are therefore difficult to be reliably modeled via white-box approaches.

We have also presented validation data for a case study where the simulation output by the framework has been compared with real data related to the execution of an open source data grid system, namely Infinispan by JBoss/Red-Hat, deployed on both a private and a public cloud infrastructure. The validation study has been based on large scale deploys on top of up to 140 Virtual Machines, and on the usage of the YCSB benchmark by Yahoo, in different configurations, as the generator of the workload profiles for our test-cases. By the data, the accuracy of the simulations in estimating core parameters such as the system throughput has been on the order of at least 80%, and on the order of 95% on the average, for all the tested configurations. Also, a case study where the framework is exploited for on-line capacity planning and reconfiguration of the data grid system is presented. Finally, the framework has been released as an open source package available to the community.

References

[1] JBoss Infinispan, Infinispan Cache Mode, 2011. <http://www.jboss.org/infinispan>.
[2] WMware, VMware vFabric GemFire. <http://www.vmware.com/products/application-platform/vfabric-gemfire/overview.html>.
[3] Oracle, Orache Coherence, 2011. <http://www.oracle.com/technetwork/middleware/co-herence/overview/index.html>.
[4] A. Lakshman, P. Malik, Cassandra: a decentralized structured storage system, SIGOPS Oper. Syst. Rev. 44 (2010).

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011

[5] D. Didona, P. Romano, S. Peluso, F. Quaglia, Transactional auto scaler: elastic scaling of replicated in-memory transactional data grids, ACM Trans. Auton. Adapt. Syst. 9 (2) (2014) 11.
[6] D. Didona, F. Quaglia, P. Romano, E. Torre, Enhancing performance prediction robustness by combining analytical modeling and machine learning, in: Sixth ACM International Conference on Performance Engineering, ICPE 2015, Austin, Texas, USA, 2015, pp. 145–156.
[7] C. Stewart, A. Chakrabarti, R. Griffith, Zoolander: efficiently meeting very strict, low-latency SLOs, in: Proceedings of the 10th International Conference on Autonomic Computing, ICAC 2013, USENIX, San Jose, CA, 2013, pp. 265–277.
[8] M. Couceiro, P. Romano, L. Rodrigues, A machine learning approach to performance prediction of total order broadcast protocols, in: Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2010, Budapest, Hungary, 2010, pp. 184–193.
[9] B. Ban, RedHat, JGroups.<http://www.jgroups.org>.
[10] B.F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, Benchmarking cloud serving systems with YCSB, in: Proc. of ACM Symposium on Cloud Computing, SoCC '10, ACM, 2010, pp. 143–154.
[11] D. Didona, P. Felber, D. Harmanci, P. Romano, J. Schenker, Identifying the optimal level of parallelism in transactional memory applications, in: Networked Systems – First International Conference, NETYS 2013, Marrakech, Morocco, May 2–4, 2013, Revised Selected Papers, 2013, pp. 233–247.
[12] D. Didona, P. Romano, Performance modelling of partially replicated in-memory transactional stores, in: Proceedings 22nd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS, IEEE Computer Society, 2014.
[13] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, M. Paleczny, Workload analysis of a large-scale key-value store, in: SIGMETRICS Perform. Eval. Rev., vol. 40(1), 2012, pp. 53–64.
[14] C. Curino, Y. Zhang, E.P.C. Jones, S. Madden, Schism: a workload-driven approach to database replication and partitioning, PVLDB 3 (1) (2010) 48–57.
[15] P. Di Sanzo, D. Rughetti, B. Ciciani, 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 '12, IEEE Computer Society, 2012.
[16] P. di Sanzo, F. Molfese, D. Rughetti, B. Ciciani, Providing transaction class-based qos in in-memory data grids via machine learning, in: IEEE 3rd Symposium on Network Cloud Computing and Applications, NCCA 2014, Rome, Italy, February 5–7, 2014, 2014, pp. 46–53.
[17] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, R. Buyya, A toolkit for modelling and simulating data Grids: an extension to GridSim, Concurr. Comput.: Pract. Exp. 20 (13) (2008) 1591–1609.
[18] S. Kounev, K. Bender, F. Brosig, N. Huber, R. Okamoto, Automated simulation-based capacity planning for enterprise data fabrics, in: 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools), 2011, pp. 27–36.
[19] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exper. 41 (1) (2011) 23–50.
[20] O. Dalle, E. Mancini, Integrated tools for the simulation analysis of peer-to-peer backup systems, in: 5th International ICST Conference on Simulation Tools and Techniques (SIMUTools), 2012, pp. 178–183.
[21] R.M. Fujimoto, W.B. Campbel, Direct execution models of processor behavior and performance, in: Winter Simulation Conference, 1987, pp. 751–758.
[22] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, L. Rodrigues, When scalability meets consistency: genuine multiversion update-serializable partial data replication, in: Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems, ICDCS, IEEE Computer Society, 2012.
[23] M.S. Ardekani, P. Sutra, M. Shapiro, Non-monotonic snapshot isolation: scalable and strong consistency for geo-replicated transactional systems, in: IEEE 32nd Symposium on Reliable Distributed Systems, SRDS 2013, Braga, Portugal, 1–3 October 2013, 2013, pp. 163–172.
[24] M.S. Ardekani, P. Sutra, M. Shapiro, N.M. Preguiça, On the scalability of snapshot isolation, in: Euro-Par 2013 Parallel Processing – 19th International Conference, Aachen, Germany, August 26–30, 2013. Proceedings, 2013, pp. 369–381.
[25] Y. Sovran, R. Power, M.K. Aguilera, J. Li, Transactional storage for geo-replicated systems, in: Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23–26, 2011, 2011, pp. 385–400.
[26] S. Peluso, P. Romano, F. Quaglia, Score: A scalable one-copy serializable partial replication protocol, in: Middleware 2012 – ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3–7, 2012. Proceedings, 2012, pp. 456–475.
[27] J.R. Quinlan, Rulequest Cubist. <http://www.rulequest.com/cubist-info.html>.
[28] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
[29] Cloud-TM: A Novel Programming Paradigm for the Cloud. <http://http://www.cloudtm.eu/>.
[30] P.A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley Longman Publishing Co., Inc., 1986.
[31] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil, A critique of ANSI SQL isolation levels, in: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, SIGMOD '95, 1995.
[32] R.B. Martin, Agile Software Development: Principles, Patterns, and Practices, Prentice Hall PTR, 2003 (ISBN: 0135974445).
[33] B. Ciciani, D. Didona, P. di Sanzo, R. Palmieri, S. Peluso, F. Quaglia, P. Romano, Automated workload characterization in cloud-based transactional data grids, in: Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012, pp. 1525–1533.
[34] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I.M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cceres, W. Emmerich, F. Galn, The reservoir model and architecture for open federated cloud computing, IBM J. Res. Develop. (2009).
[35] G. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo, S. Smallen, W. Smith, A. Grimshaw, FutureGrid – a reconfigurable testbed for Cloud, HPC, and Grid computing, in: J. Vetter (Ed.), Contemporary High Performance Computing: From Petascale toward Exascale, Chapman & Hall, 2013.
[36] Amazon EC2. <http://aws.amazon.com/ec2/>.
[37] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R.P. Abstract, Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web, in: Proc. 29th ACM Symposium on Theory of Computing (STOC), 1997, pp. 654–663.
[38] M.K. Aguilera, A. Merchant, M.A. Shah, A.C. Veitch, C.T. Karamanolis, Sinfonia: a new paradigm for building scalable distributed systems, ACM Trans. Comput. Syst. 27 (3) (2009).

Please cite this article in press as: P. Di Sanzo et al., A flexible framework for accurate simulation of cloud in-memory data stores, Simulat. Modell. Pract. Theory (2015), http://dx.doi.org/10.1016/j.simpat.2015.05.011