2012 IEEE Second Symposium on Network Cloud Computing and Applications

Auto-tuning of Cloud-based In-memory Transactional Data Grids via Machine Learning

Pierangelo Di Sanzo, Diego Rughetti, Bruno Ciciani and Francesco Quaglia
DIAG, Sapienza University, Rome, Italy

***Abstract*—In-memory transactional data grids have revealed extremely suited for cloud based environments, given that they well fit elasticity requirements imposed by the pay-as-you-go cost model. Particularly, the non-reliance on stable storage devices simplifies dynamic resize of these platforms, which typically only involves setting up (or shutting down) some data-cache instance. On the other hand, defining the well suited amount of cache servers to be deployed, and the degree of replication of slices of data, in order to optimize reliability/availability and performance tradeoffs, is far from being a trivial task. As a example, scaling up/down the size of the underlying infrastructure might give rise to scarcely predictable secondary effects on the side of the synchronization protocol adopted to guarantee data consistency while supporting transactional accesses. In this paper we investigate on the usage of machine learning approaches with the aim at providing a means for automatically tuning the data grid configuration, which is achieved via dynamic selection of both the well suited amount of cache servers, and the well suited degree of replication of the data-objects. The final target is to determine configurations that are able to guarantee specific throughput or latency values (such as those established by some SLA), under some specific workload profile/intensity, while minimizing at the same time the cost for the cloud infrastructure. Our proposal has been integrated within an operating environment relying on the well known Infinispan data grid, namely a mainstream open source product by the Red Had JBoss division. Some experimental data are also provided supporting the effectiveness of our proposal, which have been achieved by deploying the data platform on top of Amazon EC2.**

I. Introduction

With the advent of cloud computing, we have experienced 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, VMware vFabric GemFire [25], Oracle Coherence [16] and Apache Cassandra [13]. These platforms well meet the elasticity requirements imposed by the pay-as-you-go cost model. This is done by:

(i) relying on a simplified key-value data model (as opposed to the traditional relational model),
(ii) employing efficient in-memory replication mechanisms to achieve data durability (as opposed to disk-based logging) and
(iii) offering facilities for dynamically resizing the amount of hosts within the platform.

However, one aspect that still represents a core issue to cope with is related to how to (dynamically) dimension and configure the system in order to, e.g., match a predetermined Service Level Agreement (SLA), while also minimizing operating costs related to, e.g., renting the underlying virtualized infrastructure. In particular, forecasting the scalability trends of real-life, complex applications deployed on distributed transactional platforms is an extremely challenging task. In fact, as also shown in [7], when the number of nodes in the system grows, the performance of these platforms exhibits 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.

In this article we investigate on the usage of machine leaning techniques, particularly neural networks, in order to provide supports for auto-tuning in-memory transactional data grids in cloud environments. This is achieved by determining the well suited number of cache servers to be deployed ([1]), and the well suited replication degree of the data-objects maintained by the platform. Specifically, our approach allows automatic determination of system configurations that are able to satisfy some SLA, expressed in terms of predetermined throughput and latency values for a given application, possibly exhibiting variable workload intensity (e.g. in terms of number of concurrent clients), while minimizing at the same time the costs associated with the virtualized infrastructure.

While machine learning has already been exploited in the context of system tuning and optimization, to the best of our knowledge its employment in the context of tuning of in-memory data grids has been limited to capture contention on physical resources, thus requiring to be complemented via white-box analytical approaches in order to express performance forecasts by also keeping into account contention on logical resources, namely data contention (see, e.g., [7]). Further, it has been exploited limitedly to the context of fully

This work has been partially supported by the Cloud-TM project (co-financed by the European Commission through the contract no. 57784) and by COST Action IC1001 EuroTM.

[1]In conventional data grid terminology, a cache server is one server instance maintaining a slice of the data hosted by the whole platform.

978-0-7695-4943-9/12 $26.00 © 2012 IEEE
DOI 10.1109/NCCA.2012.20

replicated systems, where each cache server keeps a copy of the entire data set. In this article we take the different approach where machine learning is used to predict the system performance according to a pure black-box approach, where no integration via analytical modeling is requested. This gives rise to a highly general solution, not requiring knowledge of the inner logic (e.g. concurrency control logic) characterizing the data grid. Also, we cope with the general case of partial data replication, where each cache server is allowed to host a copy of a portion of the entire data set, which is recognized as mandatory for the achievement of significant scalability levels.

We have implemented our proposal by integrating it with the open source Infinispan data grid platform [11], namely a product by Red Had JBoss division, which represents the mainstream in-memory data repository for applications running on top of the largely diffused JBoss application server. Also, we have carried out a preliminary experimentation of our proposal by relying on the Amazon EC2 public cloud platform, whose outcoming results support the effectiveness of the provided approach.

The remainder of this article is structured as follows. In Section II we discuss related work. A brief recall on machine learning, focused on neural networks, is provided in Section III. The provided approach is illustrated in Section IV. The experimental study is presented in Section V.

II. Related Work

In the context of transactional data platforms in cloud environments, machine learning techniques have been employed by several proposals aimed at dynamically resizing the amount of back-end database servers in multi-tier environments [4], [8], [21], [26]. Typical objectives of the resize entail both performance optimization and energy consumption reduction. Compared to these proposals, we target in-memory data platforms, as opposed to traditional database technology which relies on stable storage. Also, in the above solutions the performance optimization target mostly deals with the processing of complex queries, as for the case of, e.g., generation of dynamic Web contents via reliance on relational data. Instead, our approach is not biased towards read-only accesses, thus resulting more general. Further, the proposals in [4], [8], [21], [26] deal with full replication, while we cope with the more general and scalable case of partial replication.

Control theory techniques are at the basis of several works in the area of self-tuning of application performance. These solutions often assume a linear performance model, which is possibly updated adaptively as the system moves from one operating point to another. For example, first-order autoregressive models are used to manage CPU allocation for Web servers [24]. Linear multi-input-multi-output (MIMO) models have been applied to manage different kinds of resources in multi-tier applications [17], as well as to allocate the CPU resource for minimizing the interference between VMs deployed on the same physical node [15]. Compared to these adaptive linear models, our machine learning based approach is expected to accurately capture the system behavior even in presence of non-linearities, and to allow optimized resource allocation over the entire operating space.

Less closely related approaches to our one can be found in [10], [23], where machine learning schemes are used to support automated resource provisioning in the context of non-transactional applications, such as for the case of MAP reduce applications (see [10]) or in the context of virtualized infrastructure management, such as for the case of additiona/deletion of VMs hosting generic applications (see [23]).

The closest work to our approach is the one presented in [7], where a mixed methodology based on both analytical and machine learning predictors is provided with the aim at auto-tuning in-memory transactional data grid systems. However, as pointed out before, this approach requires knowledge of specific data management algorithms (e.g. the concurrency control algorithm adopted by the data platform) in order to provide reliable predictions, while we rely on a pure black-box approach where no knowledge about the internals of the in-memory data platform is requested. Further, the work in [7] is tailored to the case of full data replication, while our proposal is able to predict performance and auto-tune the system configuration when also considering partial data replication schemes.

III. Neural Networks Recall

A neural network is a machine learning method [14] providing the ability to approximate various kinds of functions, including real-valued ones. Inspired to the neural structure of the human brain, a neural network consists of a set of interconnected processing elements which cooperate to compute a specific function, so that, provided a given input, the neural network can be used to calculate the output of the function. By relying on a learning algorithm, the neural network can be trained to approximate an unknown function f exploiting a data set $\{(\mathbf{i}, \mathbf{o})\}$ (training set), which is assumed to be a statistical representation of the function f such that, for each element $(\mathbf{i}, \mathbf{o})$, $\mathbf{o} = f\{\mathbf{i}\} + \delta$, where δ is a random variable (also said *noise*).

IV. System Architecture and Performance Prediction Scheme

A. Model of the Data Grid Platform

We consider a target data grid architecture that can be modeled by the scheme provided in Figure 1. Particularly, the system is composed by three types of entities, namely:

- cache servers, which are in charge of maintaining copies of entire, or partial, data sets;

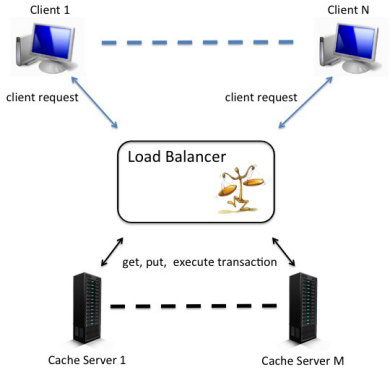

Figure 1. Reference architecture.

- a load balancer, which is in charge of allowing well balanced distribution of the overall workload across the cache servers;
- clients, which issue transactional data access and/or manipulation requests ultimately intercepted by the load balancer, and then redirected towards the appropriate cache server.

We note that such an architectural model well maps onto cloud based real system implementations, thus being a realistic one. As an example, the load balancing facility is already natively offered by public cloud providers, such as Amazon EC2.

The cache servers run proper distributed protocols in order to guarantee specific levels of isolation and data consistency while supporting transactional data accesses. For instance, the two-phase-commit 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., [11]).

Also, in our model an individual transactional request 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.

The following notations are adopted in order to define the target machine learning functions, and to describe the auto-tuning process of the data grid platform:

- N, which denotes the number of clients.
- M, which denotes the number of cache servers operating within the platform.
- G, which denotes the adopted replication degree, namely the number of copies of each data-object that are distributed across the M cache servers. In our approach we allow variations of G such that $1 \leq G \leq M$, hence capturing scenarios entailing either full data replication schemes, or partial ones. Also, G is the unique parameter describing aspects specifically related to data replication, which again provides independence of our approach of specific implementation details related to how the replicas of each individual data-object are deployed onto the cache server (such as when deploying replicated data according to consistent hashing schemes [12]).
- rt, which denotes the response time.
- thr, which denotes the system throughput.

B. Objective Functions

The rationale behind our proposal is to exploit statistics collected while the system operates in order to estimate the following two functions, representing the fulcrum of performance prediction:

$$thr = f(N, M, G) \tag{1}$$

$$rt = g(N, M, G) \tag{2}$$

where thr expresses the expected system throughput, while rt expresses the expected system response time. We note that, given the possibility of non-linearity (e.g. due to the effects of inter cache-server synchronization protocols adopted to maintain data consistency within the distributed/replicated system), it is difficult to a-priori hypothesize whether keeping, e.g., the number of clients N fixed, the throughput or the response time decreases while increasing the amount of cache servers M.

The goal of our neural network based prediction scheme is to provide approximations f_A and g_A of the functions f and g. To this purpose, we collect a set of sample data used to train the neural network, derived by observing the system behavior during a training time interval. A training sample includes the following quantities, each one expressing the mean value over a given number of subsequent data access operations (we use the apex k to indicate that they are related to the training phase):

- the set of input parameter values for f and g, i.e., N^k, M^k and G^k;
- the average system throughput thr^k;
- the average system response time rt^k.

Hence, once fixed the execution profile for the application hosted on top of the data grid platform, a training sample $(\mathbf{i}, \mathbf{o})$ is such that $\mathbf{i} = (N^k, M^k, G^k)$ and $\mathbf{o} = (thr^k, rt^k)$.

C. Platform Reconfiguration

Actual platform reconfiguration during operating phases needs to be carried out according to specific rules actuated by a proper controller, which takes in input combinations involving a (sub)set of the below listed constraints:

C1 maximum expected value for the response time;

C2 maximum sustainable cost for the infrastructure (e.g. expresses in terms of the maximum number of virtual machines to be ranted to deploy different cache server instances);

C3 minimum expected value for the system throughput;
C4 minimum value for the degree of replication of data-objects, which translates into defining the maximum degree of resilience to failures of individual cache servers ultimately causing data loss ([2]).

Given a target combination of the above constraints passed in input to the controller, such as (**C2** AND **C4**), based on the functions f_A and g_A estimated via the neural network, the controller actuates a system reconfiguration aimed at satisfying the target constraints' combination, while also minimizing or maximizing metrics not explicitly included within the target combination. This is done by exploring the output values provided by f_A and g_A over an admissible input domain, as defined by the constraints. As an example, once fixed the maximum sustainable cost, and the minimum degree of replication of individual data-objects, the controller will determine configurations that minimize the expected response time or maximize the throughput.

One important aspect for constraint **C4** relates to the supports provided by current conventional data grid systems in terms of management of the replication degree of the data-objects. Specifically, most data grid products do not support dynamic reconfiguration of the degree of data replication G, which is instead considered in our machine learning scheme, and represents one parameter that is object of dynamic tuning by our controller. As we will show in the experimental study provided in the next section, having the possibility to dynamically control the replication degree G represents a relevant facility. In fact, it infers higher flexibility to the data grid platform in terms of its ability to actually satisfy the target constraints. On the other hand, the approach we provide is highly general, thus being applicable also in contexts where G cannot be dynamically varied. The only impact on the whole approach consists in excluding G as input parameter to the functions f and g, thus leading to the exclusion of G^k values as instances for the corresponding training parameter.

V. Experimentation

We have built an actual implementation of our architectural proposal by relying on the Infinispan data grid platform, and have carried out an experimentation on top of virtualized infrastructures offered by Amazon EC2. In this section we first provide an overview of Infinispan, then we present the experimental test bed that has been used, including the description of the adopted benchmarks. Finally, the actual results are presented an discussed.

A. Infinispan Overview

Infinispan is a popular open source in-memory data grid, which is developed by JBoss/Red Hat. Currently, it represents both the reference data platform and the clustering technology for JBoss, which is the mainstream open source J2EE application server.

Infinispan exposes a key-value store data model (NoSQL), and maintains data entirely in-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 [2], but only guarantees the Repeatable Read ANSI/ISO isolation level [1] ([3]). More in detail, Infinispan implements a non-serializable variant of the multi-version concurrency control algorithm, which never blocks or aborts a transaction upon a read operation, and relies on an encounter-time locking strategy to detect write-write conflicts. Write locks are first acquired locally during the transaction execution phase, which does not entail any interaction with remote cache servers. At commit time, Two Phase Commit (2PC) [2] is executed. During the first phase (also called prepare phase), lock acquisition is attempted at all the cache servers keeping copies of the data-objects to be updated, in order to detect conflicts with transactions concurrently executing on other cache servers, as well as for guaranteeing transaction atomicity. If the lock acquisition phase is successful on all nodes, the transaction originator broadcasts a commit message, in order to apply the modifications on the remote cache servers, and then commits locally.

In presence of conflicting concurrent transactions, the lock acquisition phase (taking place either during the local transaction execution or during the prepare phase) may fail due to the occurrence of (possibly distributed) deadlocks. Deadlocks are detected using a simple, user-tunable, timeout based approach. In our experimental assessment, we consider the scenario in which the timeout on deadlock detection is set to the value zero, which is a typical approach for state of the art in-memory transactional platforms [6] to achieve deadlock freedom. In fact, distributed deadlocks represent a major threat to system scalability, as highlighted by the seminal work in [9]. We anyway remark that our machine learning based method, being black-box, is independent of specific internal configurations for the data grid, including those related to deadlock detection and resolution.

Originally, the Infinispan platform offered facilities for dynamically varying the number of cache servers, with no possibility to vary the degree of replication of individual data-objects, which was therefore statically defined at start-up time. The actual supports for keeping the degree of

[2]Recall that, in the target architecture, data-objects are supposed to be stored exclusively into main memory, thus representing volatile content that gets lost in the event of failure of the cache server where they reside onto.

[3]Improvements of the isolation level towards Update-Serializability and Snapshot-Isolation have been presented and evaluated within in-progress versions in the context of the Cloud-TM research project - see [18]. However, these optimizations have not yet been included within the product main-line, which is the reason for not considering the associated prototypes, and focusing the experimentation on the evaluation of the effectiveness of machine learning when employed in combination with current off-the-shelf versions of Infinispan.

replication constant were based on a complete reshuffle of the copies of the data-objects across the cache servers in the event of server join/leave (e.g. in the event of server crash/resume). Recent versions of this platform (namely, version 5.2.0.Beta2) have included the possibility to dynamically change the degree of replication of data-objects, which makes Infinispan perfectly match the general model provided in Section IV-B, where the degree of replication G is explicitly considered as a tunable parameter within the machine learning based reconfiguration process. This has been done via the introduction of an optimized state transfer protocol which is able to create new copies of individual data-objects (e.g. in order to put these copies on a newly joining cache server), without the need for a complete re-mapping of the whole set of objects across the platform.

B. Experimental Settings

To carry out the experimentation, we have decided to rely on a configurable synthetic benchmark which we have explicitly developed on top of the Infinispan data grid. The choice towards an early experimentation based on synthetic workloads, rather than common benchmarks, is based on the idea of widening workload configurability. In fact, known benchmarks such as TPC-C [22] only entails rigid transaction profiles. We plan anyway to carry out an experimentation with known benchmarks, such as YCSB (Yahoo! Cloud Serving Benchmark) [5] or TPC-C, as a future work.

In our synthetic benchmark application, the in-memory distributed cache has been populated with a set of 10000 data-objects, each one being associated with:

- its key, which is used to identify the object;
- its payload, with variable size between 1 and 1024 bytes.

A client executes an interleaving of put and get operations on the whole set of data-objects hosted by the distributed cache. The ratio between the number of put and get operations is established using a fixed probability value, which can be configured at start-up time. Each put operation modifies a single data-object, while each get operation 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 onto the data-objects are uniformly distributed.

As for the implementation of the neural networks used to estimate response time and throughput values, we relied on fully connected networks, with three layers each. To train the networks we used a back-propagation algorithm [20], [3], [19]. 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.

Our synthetic benchmark application has been deployed onto an Amazon EC2 cloud environment where a maximum of 20 clients, and a maximum of 6 severs have been included. We note that in our experimentation, each client mimics the behavior of a front-end server, which accesses the in-memory data layer hosted by the back-end 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 off-line training of the networks has been performed using 600 samples collected during the execution of a set of runs of the benchmark carried-out by randomly varying the number of clients, the number of server nodes and the degree of replication of the data-objects. Each sample has been obtained by averaging the values of 2000 subsequently executed operations. Then, the trained networks have been used at run-time for the estimation of throughput and response time, for all the possible valid combinations of number of servers and degree of data-object replication. Specifically, after having observed the current number of clients, the controller exploited the neural network to periodically identify the best configuration (in terms of number of servers and data replication degree), according to the given constraints, to be actuated.

As an example, in Figure 2 the system throughput and the average response time, as calculated by the trained networks for a test case we refer to as *Scenario C*, which will be presented in detail in the next section, are depicted for the case of replication degree equal to 1.

Clients and servers were deployed onto *small EC2 instances* equipped with 1.7 GB of memory 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.

C. Results

We report in this section the experimental results achieved when considering three different scenarios in term of configuration of the synthetic benchmark application, and in terms of constraints imposed to the neural network based controller. In order to observe how the neural network based-controller reacts with respect to changes of the number of clients, in all the scenarios we varied such a number over time according to the sequence of values 1,5,9,13,17,13,9,5.

1) Scenario A: In this scenario we consider a configuration of the benchmark giving rise to limited data contention across concurrent accesses. In particular, we set the benchmark parameters in order to achieve 2% of update operations (i.e. put operations). For this scenario, we impose a constraint on the cost of the infrastructure, hence relying

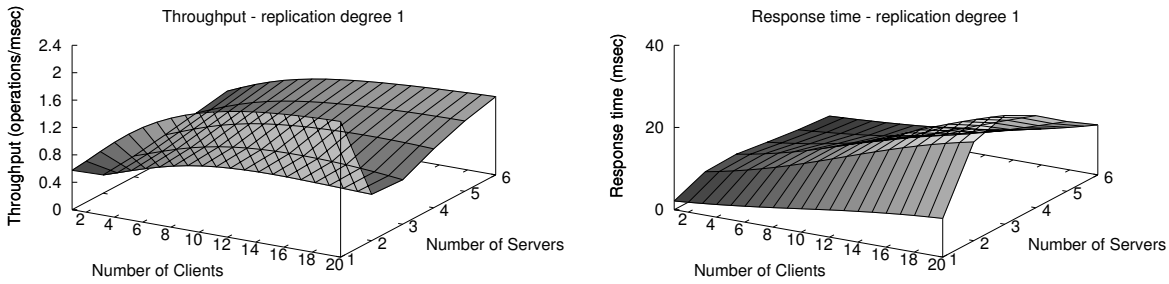

Figure 2. Example outputs of trained neural networks.

on **C2**, while also targeting the maximization of the metric associated with **C3**, namely the system throughput. **C2** is actuated by allowing a maximum cost as imposed by fixing the maximum amount of cache serves within the infrastructure to 4.

The results for this test are shown in Figure 3, where we report how the controller adjusts the actual number of cache servers to be used, and the 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 choices actuated by the controller, 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 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 neural network based controller). By the data, we get that the controller always provides dynamically tuned configurations (while varying the number of clients) which are aligned with the corresponding optimal static configurations when considering whichever fixed amount of clients in the interval between 1 and 17.

2) Scenario B: In this scenario we consider again a configuration of the benchmark giving rise to limited data contention across concurrent accesses, by still generating 2% of update operations. On the other hand, we impose a constraint on the cost of the infrastructure, by again relying on **C2**, while also targeting the minimization of the metric associated with **C1**, namely the response time. At the same time, we impose a minimum degree of replication of data-objects, as proper of **C4**. Specifically, the controller operates in order to satisfy (**C2** AND **C4**), while optimizing the response time. This time the maximum cost sustainable is expressed in terms of maximum number of deployed cache servers set to the value 3, while the minimum replication degree to be guaranteed has been set to the value 2. The results for this test are shown in Figure 4, where we report the same curves as for the previous scenario, except for the throughput curve, since in this scenario we are interested in the response time, which is shown in the top-left graph (right vertical axis). On the right of the same figure, we report the response time we measured using static configurations, such as for the throughput in the previous scenario. Also in this scenario, the controller always actuates the optimal system configuration while varying the number of clients, except for the case with 5 clients, where the controller decided for 2 servers. By the results achieved with the static configuration, we can see that the minimum response time for 5 clients is achieved with 4 servers. However, response time values achieved with 2 and 4 servers are very close to each other. Hence, the non-optimal choice is likely related to the variance of measurements at both training-time and run-time. Finally, in this scenario we note that, by the results achieved with the static configurations, the minimum response time with 1 client is achieved with 1 server. However, due to the setting of constraint **C4** (i.e. minimum replication degree equal to 2), the controller decides for using 2 servers.

3) Scenario C: In this scenario we move to a configuration of the benchmark application entailing higher contention, by generating 20% of update operations. For this scenario, we impose again a constraint on the cost of the infrastructure, hence relying on **C2**. Jointly, we impose a constraint on the degree of replication of the data-objects, as expressed by **C4** (hence generating an actual constraint in the form (**C2** AND **C4**)), while targeting the maximization of the metric associated with **C3**, namely the system throughput. As for **C2**, we admit a maximum cost in terms of maximum number of cache servers to be deployed which has been set to the value 6. On the other hand, the minimum degree of replication has been set to the value 2. Also in this case we observe that the controller always

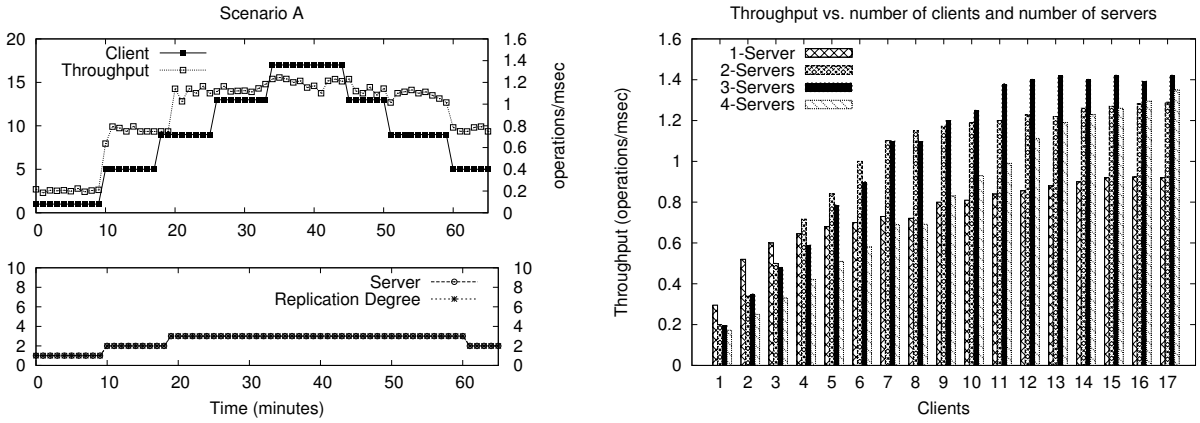

Figure 3. Results for Scenario A.

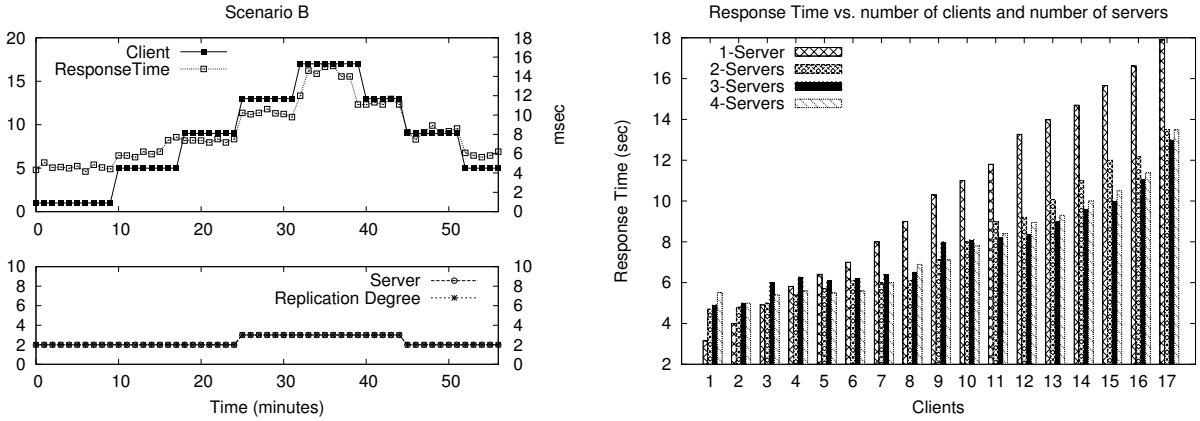

Figure 4. Results for Scenario B.

selects the best configuration, which maximizes the system throughput while satisfying the constraints (due to space limitation, we omit to explicitly show the results achieved with the static configurations).

VI. Conclusions

In this article we have shown how pure machine learning (black-box) schemes, in particular neural networks, can be used in order to dynamically optimize run-time parameters proper of transactional data grids. Specifically, with our approach, the whole data platform can be automatically reconfigured in terms of number of cache servers deployed on top of the virtualized underlying infrastructure, and degree of replication of the data-objects, in order to maximize/minimize specific performance/cost metrics. We have carried out a preliminary experimentation based on a synthetic benchmark and a real system implementation relying on the Infinispan data grid platform, which has been run on top of Amazon EC2 virtual nodes. By the results, the machine learning based approach results effective, thus opening the possibility for its employment within in-production systems. Future work along this direction entails evaluating the system with data-grid benchmarks and with more traditional benchmarks (tailored for relational data models), once set up their porting onto the key-value data model properly offered by data grid platforms.

References

[1] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and 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.

[2] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publishing Co., Inc., 1986.

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

[4] J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content web servers.

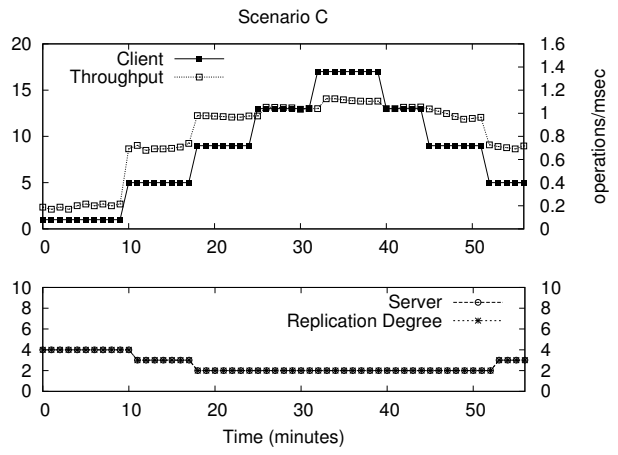

Figure 5. Results for Scenario C.

In *Proceedings of the International Conference on Autonomic Computing*, ICAC '06, pages 231–242, Washington, DC, USA, 2006. IEEE Computer Society.

[5] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proc. of ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154. ACM, 2010.

[6] D. Dice, O. Shalev, and N. Shavit. Transactional locking ii. In *In Proc. of the 20th Intl. Symp. on Distributed Computing*, 2006.

[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] S. Ghanbari, G. Soundararajan, J. Chen, and C. Amza. Adaptive learning of metric correlations for temperature-aware database provisioning. In *Proceedings of the International Conference on Autonomic Computing*, ICAC '07, pages 26–, Washington, DC, USA, 2007. IEEE Computer Society.

[9] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, 1996.

[10] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, 2011.

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

[12] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM Symposium on Theory of Computing*, STOC '97, 1997.

[13] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44, 2010.

[14] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[15] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, 2010.

[16] Oracle. Orache Coherence. http://www.oracle.com/technetwork/middleware/coherence/overview/index.html, 2011.

[17] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, 2009.

[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*, 455–465, 2012.

[19] D. E. Rumelhart and R. J. W. Geoffrey E. Hinton. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[20] 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.

[21] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy. Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proceedings of the International Conference on Autonomic Computing*, ICAC '10, pages 21–30, New York, NY, USA, 2010. ACM.

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

[23] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. A. B. Fortes. Fuzzy modeling based resource management for virtualized database systems. In *MASCOTS*, 2011.

[24] Z. Wang, X. Zhu, and S. Singhal. Utilization and slo-based control for dynamic sizing of resource partitions. In *DSOM*, 2005.

[25] WMware. VMware vFabric GemFire. http://www.vmware.com/products/application-platform/vfabric-gemfire/overview.html.

[26] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. HacigümüŞ. Activesla: a profit-oriented admission control framework for database-as-a-service providers. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, New York, NY, USA, 2011. ACM.