TIRAMOLA: Elastic NoSQL Provisioning Through a Cloud Management Platform

Ioannis Konstantinou
Computing Systems Lab
National Technical University of Athens, Greece
ikons@cslab.ece.ntua.gr

Evangelos Angelou
Computing Systems Lab
National Technical University of Athens, Greece
eangelou@cslab.ece.ntua.gr

Dimitrios Tsoumakos
Department of Informatics
Ionian University
Corfu, Greece
dtsouma@ionio.gr

Christina Boumpouka
Computing Systems Lab
National Technical University of Athens, Greece
christina@cslab.ece.ntua.gr

Nectarios Koziris
Computing Systems Lab
National Technical University of Athens, Greece
nkoziris@cslab.ece.ntua.gr

Spyros Sioutas
Department of Informatics
Ionian University
Corfu, Greece
sioutas@ionio.gr

ABSTRACT

NoSQL databases focus on analytical processing of large scale datasets, offering increased scalability over commodity hardware. One of their strongest features is elasticity, which allows for fairly portioned premiums and high-quality performance. Yet, the process of adaptive expansion and contraction of resources usually involves a lot of manual effort, often requiring the definition of the conditions for scaling up or down to be provided by the users. To date, there exists no open-source system for automatic resizing of NoSQL clusters. In this demonstration, we present *TIRAMOLA*, a modular, cloud-enabled framework for monitoring and adaptively resizing NoSQL clusters. Our system incorporates a decision-making module which allows for optimal cluster resize actions in order to maximize any quantifiable reward function provided together with life-long adaptation to workload or infrastructural changes. The audience will be able to initiate HBase clusters of various sizes and apply varying workloads through multiple YCSB clients. The attendees will be able to watch, in real-time, the system perform automatic VM additions and removals as well as how cluster performance metrics change relative to the optimization parameters of their choice.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Distributed databases*

Keywords

Elasticity, NoSQL, Automatic Cluster Resize, Markov Decision Process, Cloud Monitoring, Open-source

SIGMOD '12, May 20–24, 2012, Scottsdale, Arizona, USA.

1. INTRODUCTION

Computational and storage requirements of applications such as web analytics, business intelligence and social networking over tera- (or even peta-) byte datasets have pushed SQL-like centralized databases to their limits [12]. This led to the development of horizontally scalable, distributed non-relational data stores, called NoSQL databases (e.g., [6, 18, 8, 9], etc). NoSQL systems exhibit the ability to store and index arbitrarily big data sets while enabling a large amount of concurrent user requests. They are perfect candidates for cloud platforms that provide infrastructure as a service (IaaS) such as Amazon's EC2 [2] or its open-source alternatives (e.g.,[22, 7]): NoSQL administrators can utilize the cloud API to throttle the number of acquired resources (i.e., number of virtual machines – VMs and storage space) according to application needs.

This is highly-compatible with NoSQL stores: Scalability in processing big data is possible through *elasticity* and *sharding*. The former refers to the ability to expand or contract dedicated resources in order to meet the exact demand. The latter refers to the horizontal partitioning over a shared-nothing architecture that enables scalable load processing. It is obvious that these two properties (henceforth jointly referred to as *elasticity*) are intertwined: as computing resources grow and shrink, data partitioning must be done in such a way that no loss occurs and the right amount of replication is conserved.

Many NoSQL systems claim to offer adaptive elasticity according to the number of participant commodity nodes. Nevertheless, the "throttling" is usually performed manually, making availability problems due to unanticipated high loads not infrequent (e.g., the recent Foursquare outage [15]). Adaptive frameworks are offered by major cloud vendors as a service through their infrastructure: Amazon's SimpleDB [3], Google's AppEngine [5] and Microsoft's SQL Azure [10] are proprietary systems provided through a simple REST interface offering (virtually) unlimited processing power and storage. However, these services run on dedicated servers (i.e., no elasticity from the vendor's point of view), their internal design and architecture is not publicly documented, their cost is sometimes prohibitive and their performance is questionable [17].

A number of recent works has provided interesting insights

over the performance and processing characteristics of various analytics platforms (e.g., [17, 23, 13]), without dealing with elasticity in virtualized resources, which is the typical case in cloud environments. The studies presented in [14, 20, 26, 25] deal with this feature but do not address NoSQL databases, while [19] is file-system specific. Finally, proprietary frameworks such as Amazon's CloudWatch [1] or AzureWatch [4] do not provide a rich set of metrics and require a lot of manual labor to be applicable for NoSQL systems. Thus, although both NoSQL and cloud infrastructures are inherently elastic, there exists, to the best of our knowledge, no open-source system that combines these two technologies to offer automated NoSQL cluster resize according to dynamic workload changes.

Our work aims to bridge this gap between elasticity provisioning at different levels. Having considerable experience with NoSQL databases (including hands-on experience with the OpenStack IaaS [7]) [16], we present a distributed framework that allows (in a customizable and automated manner) virtually *any* NoSQL engine to expand or contract its resources by using a cloud management platform. The deployed system, *TIRAMOLA*, offers the following features:

- A generic VM-based control module that monitors NoSQL clusters. This module is further modified in order to report real-time, client-side statistics.
- An implementation of the decision-making module as a Markov Decision Process, enabling optimal decision-making relative both to the desired policies as well as to changes in the environment the cluster operates under.
- A real-time system that integrates these modules and utilizing popular open-source implementations for NoSQL, Cloud OS and benchmarking (HBase, OpenStack, YCSB [13] respectively) showcases its functionality: The system decides on the appropriate add/remove VM action according to the chosen cost function and relative to cluster performance.

In this demonstration, we will allow participants to interact with *TIRAMOLA* on three levels: (a) Cluster initialization: allow a choice of the initial number of VMs and size of the database, (b) Optimization function: allow a choice between different policies to be fed to the decision-making module, and (c) Workload: allow customization regarding some characteristics of the incoming load (e.g., peak amount of req/sec).

2. ARCHITECTURE

TIRAMOLA, an about 5K Python lines of code open-source project[1], features an architecture that is illustrated in Figure 1. The *Decision Making* module incorporates both the optimization function given through customer policies as well as cluster- and client-side metrics and periodically decides on cluster resize operations. It interacts with the *Cloud Management* module that contacts the cloud vendor to adjust the cluster's physical resources by releasing or acquiring more virtual machines. The *Cluster Coordinator* module executes higher level add, remove and rebalance commands according to the particular NoSQL system used. Finally, the *Monitoring* module maintains up-to-date performance metrics that collects from the cluster nodes and the clients. Below we describe each module in more detail.

[1] `http://tiramola.googlecode.com`

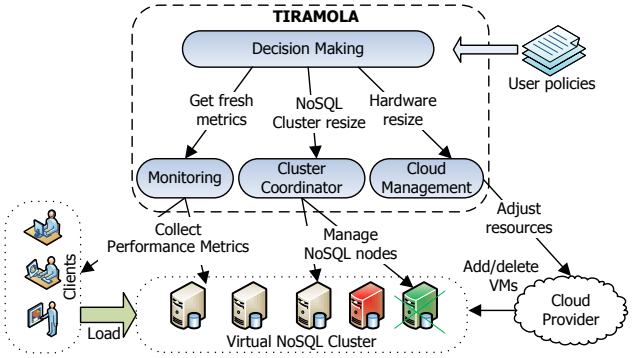

Figure 1: Architecture of our Cloud-based NoSQL elasticity-testing framework.

Monitoring Module: Our system takes a passive monitoring approach. Currently, it receives data from Ganglia [21], a scalable distributed monitoring tool that allows the user to remotely collect live or historical cluster statistics (such as CPU load averages, network, memory or disk space utilization, number of open client threads, etc) through its XML API and present them through its web front-end via real-time dynamic web pages. Apart from server side metrics, we have performed modifications to allow Ganglia to collect user-related metrics. This is necessary, since the system state may also depend on user-related information such as mean query latency and query arrival rate. To achieve this, we have modified our clients so that each one reports its own metrics by utilizing a well-known Ganglia operation called "gmetric spoofing" (we do not consider malicious clients, therefore every reported metric is assumed to be correct). With this mechanism, the monitoring module feeds the decision making module with an up-to-date system state, taking into account both client- and server-side metrics.

Cloud Management: Our system interacts with the cloud vendor using the well known *euca2ools*, an Amazon EC2-compliant, REST-based client library. The Decision Making module interacts with this module when it commands for a resize in the physical cluster resources, i.e., the number of running VMs. Our cloud management platform is a private OpenStack [7] installation. The use of euca2ools guarantees that our system can be deployed in Amazon's EC2 or in any EC2-compliant IaaS cloud. We have created an Amazon Machine Image (AMI) that contains preinstalled versions of the supported NoSQL systems along with the Ganglia monitoring tool.

Cluster Coordinator: VM coordination is done with the remote execution of shell scripts and the injection of on-the-fly created NoSQL specific configuration files to each VM. Higher level "start cluster", "add NoSQL node" and "remove NoSQL node" commands are translated in a workflow of the aforementioned primitives. Our framework ensures that each step has succeeded before moving to the next one.

Decision Making Module: This module is responsible for deciding the appropriate cluster resize action according to the applied load, cluster operating state and user-defined cost function. *TIRAMOLA* formulates this process as a Markov Decision Process (MDP) that continuously identifies the most beneficial action relative to the current system state. Benefits (both short and long-term) are defined through a *reward* function that expresses the optimization/cost-model each user wishes to operate under.

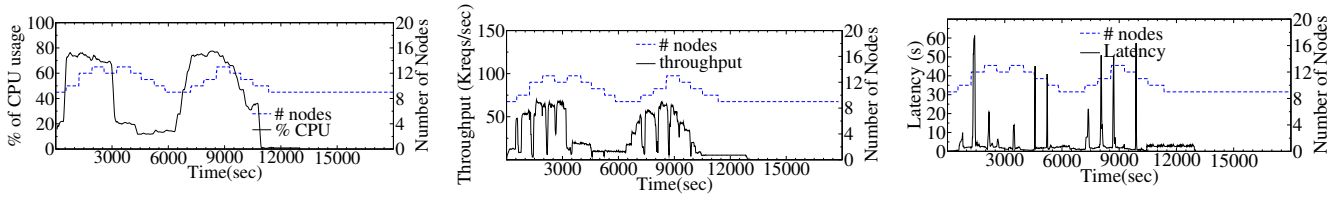

Figure 2: CPU usage, query throughput and query latency for the $r_1(s)$ reward function.

Metrics that are used to define this function come from the cluster status (e.g., CPU load, available memory, throughput, etc) and the applied workload (e.g., average query latency and arrival rate). Upon reaching to a resize decision, the module requests addition or removal of a number of VMs using the Cloud Management and Cluster Coordinator modules. For finite MDPs there exists at least one action strategy that maximizes the expected sum of rewards or, equivalently, dictates greedy state transitions according to the optimal *state value functions* $V^*(s)$ that satisfy Bellman's equation [24]. We formulate the Decision Making module as an MDP (S, A, P, γ, R) as follows:

Set of States S: $S = \{S_1, S_2, ..., S_M\}$, where S_i represents a cluster with i NoSQL worker nodes (VMs) up and running.

Actions $A(s)$: Available actions are to `add` and `remove` node or do nothing (`no-op`). The action space may also be quantified by allowing only certain number(s) of VM instances to be added or deleted (e.g., `add_2`, `add_4` and `remove_2`, `remove_4`, etc).

The transition probability matrix P: The quantity p_{ij}^a is the probability of transitioning from state i to state j given that action a was taken. In our formulation, the transition probabilities state that a transition is only permissible if adding (removing) VMs results in the exact number of VMs being added (subtracted) to the initial cluster, or after a `no-op` the state remains the same. The model can be generalized to include the possibility of a partially successful permissible operation, i.e., $p_{ij}^a = f, 0 \leq f \leq 1$ (e.g., adding 4 nodes may result in only increasing cluster size by 2).

The discount factor $0 \leq \gamma < 1$. It accounts for both present and future rewards.

Reward function R: Function $R(s)$ returns a numerical level of "goodness" of being at state s. This is an important part of the formulation, as reward maximization by the module leads to different optimal policies using different reward functions. In general, $r(s) = \varphi(\text{gains}, \text{costs})$, i.e., a user-defined cost model is set by appropriately incorporating various measurable or verifiable quantities into the reward function. For example, high throughput or served query rate can be profitable while query latency that violates an SLA, costs per running VM, etc, can be considered costly. Note here that our work does not provide an "optimal" definition of the reward function. Rather, we present a system that can optimally manage VM resources according to any function provided.

Thus, the system of equations on the *optimal* state value functions becomes: $V^*(s) = R(s) + \max_a\{\gamma V^*(s')\}$ (1) for all permissible actions a from state s. This formulation allows for a large degree of freedom in three points: The assumptions about the knowledge we have of the system, the level of customization that can be achieved and the required computation. First, the finite MDP allows for optimal solution regardless of our knowledge of the environment or its dynamics: If a strict model can be deduced, Dynamic Programming techniques over Bellman's equations can identify the optimal policy with less experimentation cost. Alternatively (which is the case for the current *TIRAMOLA* deployment), the system may learn directly from experience without a model of the environment's dynamics. Second, learning is performed in real-time and continues through the entire life of the system. The module learns and acts simultaneously, reacting to changes in the cost model, the incoming load, the infrastructure, etc. Third, the definition of the reward function leads to different optimization goals by different providers.

Solving the system of linear equations (1) requires the $R(s)$ values for all states. To estimate those values from previous experience, we utilize an incremental, online method sketched below: A multi-dimensional dataset, with the metrics participating in the reward function as the different dimensions, is constantly enriched at each polled interval (e.g., on a per-minute basis). Data points that correspond to system measurements for similar to the current stimuli (i.e., incoming load) are dynamically clustered, with centroids being used in order to compute the $R(s)$ values.

To see an example of how *TIRAMOLA* allows for adaptive cluster resize without any human involvement, we have utilized a 9-node HBase cluster loaded with 20M YCSB records. We test our system under a realistic and intensive workload, in which load increases during peak hours and drops during non-peak hours (e.g., Figure 14 in [11]). The reward function $r_1(s)$ chosen here, considers both throughput (thr) and latency (lat) besides the cost per VM: $r_1(s) = B \cdot thr - C \cdot |\text{VMs}| - D \cdot lat$, with B, C, D appropriate constants. Figure 2 shows the results that this policy produces. The decision making module increases the number of nodes to a "modest" number of 13 servers, passing from states 9-10-12-13 and 9-10-11-13 during the load increase. It uses the same transition path to return the cluster to each original state, removing one HBase node at a time. Although it incorrectly removes and re-adds a server (3000-4000 seconds into the experiment), it correctly spots the load decrease and returns the system to its original state. Similarly, it can be shown that the proposed system adaptively reassigns resources according to the reward functions presented and the incoming load.

3. DEMONSTRATION DESCRIPTION

The demonstration will allow attendees to interact with *TIRAMOLA* on three levels: Cluster initialization, user policy and imposed workload. A comprehensive real-time GUI will showcase the system's ability to setup an HBase cluster, impose various dynamic loads on it and automatically adapt the number of VMs to optimize the policy provided by the user. Users can access *TIRAMOLA* through a simple web interface, as depicted in Figure 3. The interface and demonstration consists of three parts described in detail:

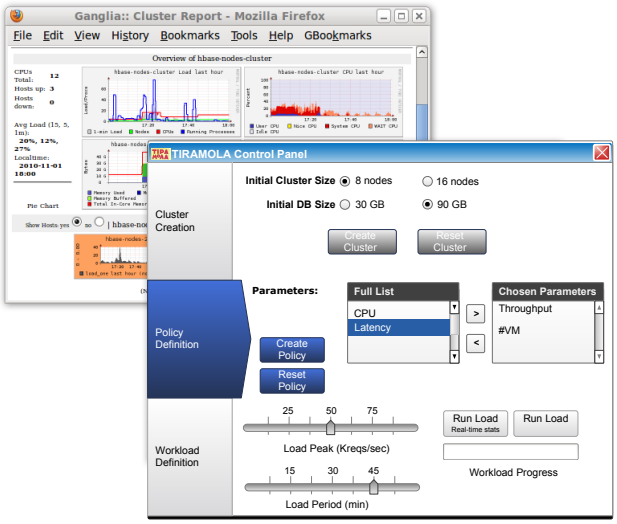

Figure 3: *TIRAMOLA* demo interface.

Cluster Initialization: Users will be able to create an initial HBase [6] cluster. The choices here will be over the initial number of nodes as well as the size of the shared data. The total number of available VMs will be 24. To allow room for both additions and removals, 8 or 16 initial nodes can be setup. Moreover, to allow for timely and interactive demos, "pre-constructed" images of two database sizes (30 GB and 90GB) will be ready to be loaded to the cluster upon user choice to avoid the time-consuming task of loading using clients. Note here that, as soon as the cluster is initiated, users will have access to the web-frontend of the virtual cluster's Ganglia monitoring tool, with real-time presentation of various performance metrics.

Policy definition: In this tab, users may choose which of the metrics will affect the behavior of the decision making module. In effect, combinations of reward functions that contain *query latency*, *cluster throughput*, *number of VMs* and *cluster CPU* can be constructed.

Workload specification: We utilize the YCSB Cloud Serving Benchmark framework [13] in order to pose load to our clusters. Through the third tab, users may alter two distinctive characteristics of a basic "skeleton" workload which periodically increases and decreases (approximating a sinusoid function): The period of change and its peak value. Thus, participants can actively control how dynamic and intense the load imposed upon the cluster is.

As soon as all these parameters have been set, a YCSB workload will be executed at the newly created cluster. Users can now watch various cluster metrics through its Ganglia frontend in real-time, the operations that *TIRAMOLA* decides and how they affect the performance from the clients' point of view. Finally, aggregate reports and graphs documenting mean query latency, served requests per second and the number of dedicated resources over time can be printed at the users' request at the end of the workload.

By providing a wide range of configurable parameters and having adequately provisioned resources to ensure timely demonstrations, our system will showcase its adaptation relative to realtime choices: More nodes added for larger load peaks; inability to add or remove nodes when certain metrics are not included in the policy; perform VM additions and removals at different rates relative to the workload period, etc. Aggregate reports created and saved at the end of each load can be used to directly compare behaviors, initiate discussions and comment over various aspects of the system with the attendees.

4. REFERENCES

[1] Amazon CloudWatch. http://aws.amazon.com/cloudwatch/.

[2] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[3] Amazon SimpleDB. http://aws.amazon.com/simpledb/.

[4] AzureWatch. http://www.paraleap.com/azurewatch.

[5] Google AppEngine. code.google.com/appengine.

[6] HBase Homepage. http://hbase.apache.org.

[7] OpenStack: The open source, open standards cloud. http://openstack.org.

[8] Project Voldemort. http://project-voldemort.com.

[9] Riak Homepage. https://wiki.basho.com/display/RIAK/Riak.

[10] Windows Azure Platform. http://www.microsoft.com/windowsazure/.

[11] A. Qureshi et al. Cutting the Electric Bill for Internet-Scale Systems. In *SIGCOMM*, 2009.

[12] D. J. Abadi. Data Management in the Cloud: Limitations and Opportunities. *IEEE Data Eng. Bull*, 32(1):3–12, 2009.

[13] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In *ACM SOCC*, pages 143–154, 2010.

[14] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero. Service Specification in Cloud Environments Based on Extensions to Open Standards. In *COMSWARE*, 2009.

[15] E. Horowitz. Foursquare Outage Post Mortem. http://bit.ly/c4GnV0.

[16] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris. On the Elasticity of NoSQL Databases over Cloud Management Platforms. In *CIKM*, 2011.

[17] D. Kossmann, T. Kraska, and S. Loesing. An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. In *SIGMOD*, pages 579–590, 2010.

[18] A. Lakshman and P. Malik. Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, 2010.

[19] H. C. Lim, S. Babu, and J. S. Chase. Automated Control for Elastic Storage. In *ICAC*, 2010.

[20] P. Marshall, K. Keahey, and T. Freeman. Elastic Site: Using Clouds to Elastically Extend Site Resources. In *CCGRID*, 2010.

[21] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7):817–840, 2004.

[22] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *CCGRID*, pages 124–131, 2009.

[23] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis. In *SIGMOD*, pages 165–178, 2009.

[24] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[25] K. Tsakalozos, H. Kllapi, E. Sitaridi, M. Roussopoulos, D. Paparas, and A. Delis. Flexible Use of Cloud Resources through Profit Maximization and Price Discrimination. In *ICDE*, 2011.

[26] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigümüs. Intelligent Management of Virtualized Resources for Database Systems in Cloud Environment. In *ICDE*, 2011.