2018 IEEE 34th International Conference on Data Engineering

Elton: a Cloud Resource Scaling-out Manager for NoSQL Databases

Athanasios Naskos [#1], Anastasios Gounaris [#1], Ioannis Konstantinou [*2]

[#] *Aristotle University of Thessaloniki, Greece*
[1] {anaskos,gounaria}@csd.auth.gr

[*] *National Technical University of Athens, Greece*
[2] ikons@cslab.ece.ntua.gr

***Abstract*—We present the Elton tool, a publicly available cloud resource elasticity management system tailored to NoSQL databases. Elton is integrated in the Ganetimgr web platform, and offers an easy to use web interface, through which monitoring and horizontal scaling of NoSQL databases can be performed and what-if analysis queries are enabled. Elton uses Markov Decision Processes (MDPs) as the underlying modeling framework, and encapsulates state-of-the-art horizontal scaling policies that offer different trade-offs between performance and monetary deployment cost. Its main novelty is that it employs probabilistic model checking to allow for both efficient elasticity decisions and analysis of scaling actions and serves as a case study about the benefits of model checking in online decision making and analysis.**

I. Introduction

A key feature of cloud environments is the ease of scaling with regards to provided resources. Users are able to add or remove system (e.g., RAM) and computing (i.e., virtual machines (VMs)) resources, known as *vertical scaling* or *scale-up* and *horizontal scaling* or *scale-out*, respectively, or move the computing resources to different physical locations, which is termed as *migration*. The adaptation of the cloud resources is referred to as *cloud elasticity*, and is typically performed in an automated manner. The cloud applications that make use of the latter feature try to lease just the appropriate amount of resources at any point to guarantee for their proper functionality without incurring unnecessary monetary cost.

Cloud-hosted databases can benefit from all types of elasticity (e.g., [1], [2], [3]). However, NoSQL databases fit better to the scaling-out paradigm, given that they are usually equipped with built-in mechanisms for dynamic expansion or shrinkage over a distributed environment. To efficiently handle the horizontal elasticity in NoSQL database systems, we need to take into consideration their special behavior. More specifically, such systems might be unpredictable, exhibiting significantly varying performance under similar external workload for the same number of VMs, which is not amenable to analytical modeling [1], [4]. Moreover, elasticity actions may be followed by significant transient periods, which, if not treated with care, may lead to destabilization of the system.

The known cluster and cloud management systems and their interfaces (e.g., `OpenStack`[1], `Amazon EC2`[2], `Ganeti`[3], `Ganetimgr`[4]) offer limited or even no elasticity handling mechanisms for NoSQL databases. Most of the proposals offer generic elasticity handling mechanisms with simple decision policies (e.g., rule-based ones). Our contribution is the presentation of a scaling-out manager tailored to NoSQL databases. The novelty of our approach is in the usage of model checking in a non-traditional manner, namely to assist in dependable online decision making.

More specifically, we build upon the state-of-the-art elasticity approach for NoSQL introduced in [4], which have been incorporated to `ganetimgr`, a web platform on top of the Ganeti manager, forming the `Elton` application[5]. We offer a range of decision policies that strike configurable balance between performance and deployment cost. Unlike other proposals that treat the elasticity handling problem as only an optimization one, we also focus on dependability through (probabilistic) model checking. We further leverage this feature to offer an analysis service so that users better understand the behavior of their NoSQL cluster and the consequences of the applied elastic actions. An example analysis query is: *"What will the probability of mean latency exceeding a threshold in the next three periods be, if no VM is added now?"* Such analyses are also employed to drive elasticity decisions. In Elton, we have developed a web interface to access our decision making mechanism and/or submit analysis queries in the form of Probabilistic Computation Tree Logic (PCTL) properties. The verification of PCTL properties is capable of supporting several what-if scenarios. In addition, we offer the capability to emulate elasticity decisions using past traces.

To showcase the Elton's functionality, we have used an Apache Cassandra NoSQL cluster deployed in a private cloud infrastructure. The workload for submitting requests is according to the YCSB benchmark[6]. In the demo scenario, we show how to conveniently set-up, enact, monitor and analyze

[1] https://www.openstack.org/
[2] https://aws.amazon.com/ec2/
[3] http://www.ganeti.org/
[4] https://github.com/grnet/ganetimgr
[5] http://interlab.csd.auth.gr/anaskos/elton
[6] https://github.com/brianfrankcooper/YCSB

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00196

elasticity. Ease of use is a main design principle, therefore, for the non-experts, pre-built PCTL properties and default values for all main configuration parameters are provided.

In Section II, we briefly present the technical background of our decision mechanism. In Section III, we present the Elton elasticity management system. We briefly discuss related work in Section IV, and we conclude in Section V.

II. Summary of Elton's Technical Details

In a nutshell, Elton is an implementation of the framework in [4], where, in each time period, a MDP model is instantiated on-the-fly. Then, PCTL properties are verified on top of these models, in a way that their results can be directly translated in elasticity decisions. In horizontal scaling, such decisions include add one or more VMs, remove one or more VMs, or keep the current configuration.

The rationale of MDP modeling is to have a distinct model state for each combination of (i) cluster configuration (abstracted by the number of VMs) and (ii) expected system behavior captured by representative query latencies. These latencies are derived through clustering past latencies for similar past conditions in terms of the volume of incoming requests kept in the log files. The state transition probabilities are commensurate to the cluster sizes. The model captures the evolution of the system for a configurable number of future periods, whereas prediction modules forecast the anticipated workload in these periods. Finally, each state is associated with a reward, which is defined according to user-defined utility functions. Both the model creation and verification are performed with the help of the `PRISM` tool [5].

The elasticity decision making is directly supported by the verification of PCTL properties on top of the MDP model. When the decision involves a change in the number of VMs, decision making is suspended to allow the system to stabilize before the next decision, whereas this suspension is captured by the MDP models. The main scenario that Elton targets is to ensure that the query latency does not exceed a user-defined threshold without paying for more resources than necessary; therefore, both over- and under-provisioning need to be avoided. The policies that Elton supports include the following (assuming that the utility functions are defined in a way that the lower the reward the better):

1) *Optimal Reward Advanced (OR_{adv})*: At a first stage, the elastic actions with the minimum reward are selected; then the action among all candidates from the first stage with the least maximum probability of latency violations is selected.
2) *Bounded Reward Simple (BR_{simple})*: The goal of this policy is to reduce the deployment cost at the expense of allowing a decrease in the reward up to a user-defined proportion of *x%*. To this end, the *remove* or *no operation* action belonging to the beginning of the strategy with the minimum sub-optimal reward (i.e. reward up to *x%* higher than the optimal reward) is selected.
3) *Bounded Reward Economy (BR_{econ})*: Similarly to the previous one, this policy chooses the *remove* action with a sub-optimal reward (i.e. not necessarily the minimum sub-optimal one) that achieves the greatest reduction in the number of VMs.
4) *Bounded Reward - Bounded Probability Economy ($BR - BP_{econ}$)*: This policy adds a quantitative verification step to the previous policy. It chooses the action with a sub-optimal reward that is going to remove the greatest number of VMs provided that its maximum probability of latency violation falls below an additional user-defined threshold.
5) *Bounded Reward - Bounded Probability Quality ($BR - BP_{qual}$)*: This policy has exactly the opposite logic compared to the previous one, as it firstly chooses the action that adds the greatest number of VMs, then it chooses the *no operation* action and, as a last option, it chooses the action, which is going to remove the least number of VMs.
6) *Bounded Reward - Bounded Probability Stability ($BR - BP_{stab}$)*: This policy tries to balance the previous two policies and make as small changes as possible. It prefers the least change in the number of VMs with a slight preference in the additions.

The policies above are further detailed and evaluated in [3], [6], where it is shown that (i) they can significantly improve upon other policies, such as rule-based ones (e.g., compared to rule-based ones, under-provisioning is improved by an order of magnitude while also avoiding over-provisioning); (ii) no policy dominates each other; and (ii) they are capable of yielding a particularly wide range of trade-offs between monetary cost and meeting user-defined latency thresholds. All policies are charging model-aware: if VMs are charged per hour, no VMs are released before the current hour expires.

III. Elton Interface and Demo Scenarios

Elton's interface is designed in a way that is easy to use through hiding the aforementioned technical details and providing default configuration. After briefly presenting its design, we describe the intended demo scenarios[7].

A. Front-end

We have integrated Elton in the Ganeti Manager web platform (aka Ganetimgr). The latter is an open source project which stands on top the Ganeti virtual machine cluster management tool. It is developed in python and uses the `DJango` web framework. DJango's modular nature has simplified the development of Elton as a standalone add-on application embedded to Ganetimgr. Similarly, Elton can be integrated in Ganeti Web Manager (Ganeti-webmgr), which is akin to Ganetimgr; we have selected the latter due to its more advanced interface. We have extended Ganetimgr with three features: i) *Elasticity*, ii) *Analysis*, and iii) *Monitoring*; screenshots are presented in Figure 1.

[7]demo video at https://www.youtube.com/watch?v=UkgLs89ifzQ

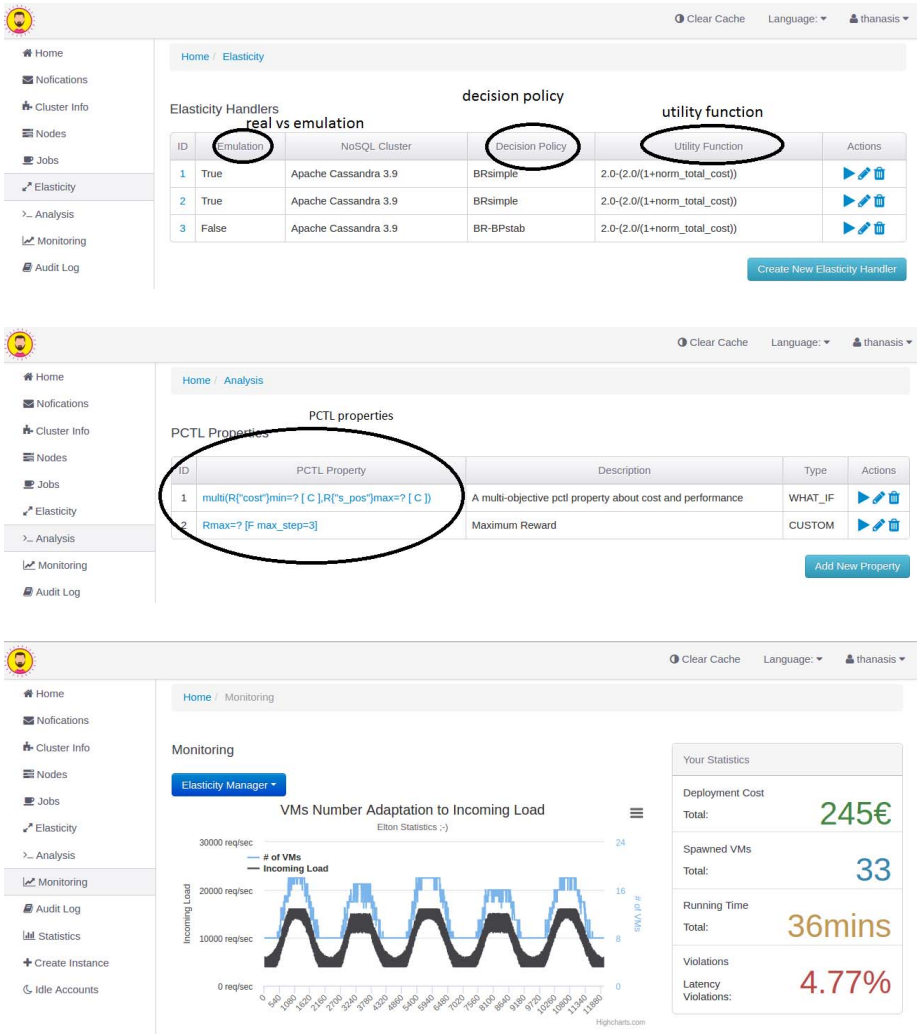

Fig. 1. Elton's Web Interface for elasticity (top), analysis (middle) and monitoring (bottom).

Considering the *Elasticity* interface, a summary of all the available elasticity configurations is displayed in the figure. The user can interact with each configuration by deleting it, editing it, or starting the corresponding NoSQL cluster and elasticity manager. Multiple clusters and elasticity managers may be deployed simultaneously. Clicking on the *"Create new Elasticity Handler"* button, the user is transferred in a form containing the available configuration for both the NoSQL cluster and the elasticity handling mechanism. In the same form, the user also selects whether to emulate the deployment or to deploy a real NoSQL cluster. In the case of emulation, the user provides system traces. Using Ganglia, Elton can be also guided to gather such traces, which will be used as training input to our decision making mechanism. Clusters of Cassandra, Hbase and Infinispan are supported; support for further NoSQL systems can be easily added. In addition, in the emulation mode, the user is able to set an imaginary load pattern (e.g. sinusoidal with spikes), which is useful for testing.

In each configuration, the decision policy is selected. Further, the user defines the utility functions that are used to assign rewards to MDP model states. We currently support three templates of utility functions for assigning state rewards: (i) reward decreases in manner that is inversely proportional to the number of VMs; (ii) reward decreases in manner that is inversely proportional to the actual deployment of VMs; and (iii) reward being a linear function of observed latency and deployment cost. In each template, if latency exceeds the user-defined threshold, a (high-value) penalty reward can be set. The pros and cons (along with configuration issues) of these templates are discussed in [6]. Finally, the manager is equipped with standard mechanisms to forecast future load (ARMA, ARIMA, or linear regression).

While an elasticity manager is running, the user can analyze the behavior of the system using one of the several pre-specified PCTL properties in the *Analysis* interface, presented in Figure 1(middle), or providing a custom PCTL property. An

option to download the corresponding instantiated MDP model in the PRISM format is also available to allow for further analysis. The analysis can refer to past points during emulation, e.g., to answer questions of the type *"what would be the maximum probability of latency violation if, at a given time point, 2 more VMs were added?"* In PCTL and in accordance to the underlying MDP models, instances of such queries are expressed as follows: *Pmax=? [F (phase=4) & (latency > latency_threshold) & (num_nodes=6) & (init_num_nodes=4)]*.

At the bottom of the figure, the visual representations of the variation of the number of VMs and the response latency in relation to the variation of the incoming load, based on the collected logs, is shown. The diagrams are also accompanied by information about the estimated current deployment cost, the proportion of latency violations, the running time and other useful measurements. The deployment cost is indicative, since we employ private machines and is computed if the deployment cost of VMs was set by Amazon EC2, e.g., assuming `c3.4xlarge` instance types.

B. Showcase Scenario

The aim of the demo is to show how a user can easily i) set-up and enact; and ii) monitor and analyze elasticity in both emulated and real settings.

During set-up, we show how to define a cluster assuming the existence of pre-build VM images containing Cassandra in Ganeti. The user can set the minimum and maximum number of VMs, along with the initial number of VMs. In both emulated and real settings, past logs (traces) are required. These are in a simple `csv` format containing the following fields: timestamp, number of vms, latency, throughput and CPU utilization (the last two are not used by our demonstrated elasticity policies). Expert users may opt to modify the default configuration of the elasticity manager changing the default values regarding the maximum number of VMs allowed to add/remove in a single step, weights in the utility functions, exact algorithm for load forecasting, and other aspects, such as the usage of a smoothing window through moving average-based techniques, whether the charging model is hourly based and MDP details (transient period length, log clusters withing states, model steps, and so on). All these settings come with pre-completed default values for non-experts.

When elasticity making is enacted, the focus will be shifted to monitoring and analysis. Given that elasticity actions may be triggered every 5 minutes or less frequently, during the demo we aim to emulate a real deployment consisting of 8-16 VMs running Cassandra and processing the YCSB benchmark. The monitoring will be performed through the interface shown in Figure 1 (bottom), which provides a concise overview of the main metrics of interest (and their evolution). Finally, analysis queries like the two examples given earlier in the text will be run during the demo.

IV. Related Work

A framework for cloud service elasticity behavior estimation and evaluation, which evaluates the impact of an elastic action to the cloud service, assisting users to define more accurate elasticity decision rules is proposed in [7]. In a more recent work [8], a solution is presented to support the development and the deployment of elastic systems. The authors propose a framework for monitoring and analyzing rule-based elastic systems, called MELA. We differ in that we build upon probabilistic model checking for both decision making and analysis, being able to verify a wider range of properties, namely those supported by PCTL, and we also depart from relying on rules to guide elasticity.

Similarly to our approach, authors in [9] apply what-if analysis. They propose a prediction framework for estimating the distribution of the response latency of a request in a cloud hosted web applications, given a hypothetical cloud resource configuration and its workload. The estimation is done using causal dependencies. Analyzing multiple what-if scenarios the user is able to determine the optimal cloud resource configuration. We differ in that we tightly couple analysis and decision making within our policies, rather than merely providing analysis functionalities.

Finally, the high-level module design of Elton is based on the Tiramola elasticity manager, which employs MDPs but not probabilistic model checking [10].

V. Conclusions

In this work, we present Elton, a cloud resource elasticity management system tailored to NoSQL databases. We have integrated Elton in Ganetimgr and presented the three added interfaces, namely i) *Elasticity*, ii) *Analysis* and iii) *Monitoring*. Finally, we have showcased the functionality of the proposed tool in both an emulated and a real environment.

References

[1] S. Das, F. Li, V. R. Narasayya, and A. C. König, "Automated demand-driven resource scaling in relational database-as-a-service," in *Proc. of SIGMOD*, 2016, pp. 1923–1934.

[2] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi, "Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration," *Proc. of VLDB*, vol. 4, no. 8, pp. 494–505, 2011.

[3] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas, "Dependable horizontal scaling based on probabilistic model checking," in *Proc. of CCGrid*, 2015, pp. 31–40.

[4] A. Naskos, A. Gounaris, H. Mouratidis, and P. Katsaros, "Online analysis of security risks in elastic cloud applications," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 26–33, 2016.

[5] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: probabilistic model checking for performance and reliability analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 40–45, 2009.

[6] A. Naskos, A. Gounaris, and P. Katsaros, "Cost-aware horizontal scaling of nosql databases using probabilistic model checking," *Cluster Computing*, vol. 20, no. 3, pp. 2687–2701, 2017.

[7] G. Copil, D. Trihinas, H. L. Truong, D. Moldovan, G. Pallis, S. Dustdar, and M. D. Dikaiakos, "ADVISE - A framework for evaluating cloud service elasticity behavior," in *Proc. of ICSOC*, 2014, pp. 275–290.

[8] G. Copil, D. Moldovan, H. L. Truong, and S. Dustdar, "Continuous elasticity: Design and operation of elastic systems," *IT - Information Technology*, vol. 58, no. 6, pp. 329–348, 2016.

[9] Y. Jiang, L. R. Sivalingam, S. Nath, and R. Govindan, "WebPerf: evaluating what-if scenarios for cloud-hosted web applications," in *Proc. of ACM SIGCOMM*, 2016, pp. 258–271.

[10] I. Konstantinou, E. Angelou, D. Tsoumakos, C. Boumpouka, N. Koziris, and S. Sioutas, "TIRAMOLA: elastic nosql provisioning through a cloud management platform," in *Proc. of SIGMOD*, 2012, pp. 725–728.