Probabilistic Model Checking at Runtime for the Provisioning of Cloud Resources

Athanasios Naskos, Emmanouela Stachtiari, Panagiotis Katsaros, and Anastasios Gounaris

Aristotle University of Thessaloniki, Greece
{anaskos,emmastac,katsaros,gounaria}@csd.auth.gr

Abstract. We elaborate on the necessary ingredients of a model-driven approach for the dynamic provisioning of cloud resources in an autonomic manner. Our solution has been experimentally evaluated using a NoSQL database cluster running on a cloud infrastructure [5]. In contrast to other techniques, which work on a best-effort basis, we can provide probabilistic guarantees for the provision of sufficient resources. Our formal verification approach is based on the probabilistic model checking of Markov Decision Processes (MDPs) at runtime. We present: (i) the specification of an appropriate MDP model for the provisioning of cloud resources, (ii) the generation of a parametric model with system-specific parameters, (iii) the dynamic instantiation of MDPs at runtime based on logged and current measurements and (iv) their verification using the PRISM model checker for the provisioning/deprovisioning of cloud resources to meet the set goals[1].

1 Introduction

This work aims to a practical model-driven solution for the following problem. We consider a cloud application, such as a web-enabled NoSQL database, running on a cluster of virtual machines (VMs). The external load of service requests submitted by end-users evolves over time. Each request has to be served within a fixed period of time; more specifically, a threshold on acceptable response latency is set. To achieve this goal, we rely on horizontal scaling *elasticity actions*, i.e., new VMs may be added on the fly to cope with load increases and VMs can be also released when the load decreases. The main challenge is to develop a decision making policy that avoids both resource under-provisioning, which leads to violations of the latency threshold, and over-provisioning, which leads to low infrastructure utilization and unnecessary economic cost.

Existing decision making policies such as the one implemented in Amazon's EC2 manager mainly work on a best-effort basis and there is no way to provide guarantees for their performance in diverse workload scenarios. Some other

[1] This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund."

model-driven proposals [6] combine Markov Decision Process (MDP) models with reinforcement learning-based policies.

Our decision making solution is based on the probabilistic model checking of dynamically instantiated MDPs by using the PRISM tool [1] at runtime. A decision step is activated periodically, e.g., every 30 seconds or every few minutes. Each decision (or elasticity) step is split into the following three phases:

1. Appropriate MDP models are dynamically instantiated, which reflect the predicted system utility and the possibility of latency threshold violations. These models are constructed based on the monitored incoming load and past log measurements of response latency, for similar incoming load values.
2. MDPs are verified online using PRISM, in order to find the optimal elasticity decisions for the solved model instances.
3. The selected elasticity action to be applied is chosen from the set of possible actions $\{$*add, remove, no_op*$\}$ that respectively correspond to adding new VMs, releasing existing VMs and leaving the cluster unchanged. If any of the first two actions are decided, then the run of the next elasticity step is suspended until the system stabilizes.

In [5], we have experimentally evaluated and compared the described approach with the mentioned alternatives. The presented results are based on traces from a real NoSQL database cluster under constantly evolving external load and they are particularly promising: we can support decision policies that improve upon the state-of-the-art in significantly decreasing under-provisioning, while avoiding over-provisioning. Here, we focus on our MDP modeling approach for the first elasticity phase, whereas details for the two other phases are provided in [5].

In Section 2, we elaborate on our approach for the specification of an appropriate MDP model. Section 3 discusses the generation of a parametric model with system-specific parameters, the dynamic instantiation of MDPs, and their verification at runtime. Finally, we conclude with a critical review on the practicality and the prospects of the proposed solution.

2 An MDP for the control of cloud resource provisioning

An MDP allows to capture both the non-deterministic and the probabilistic aspects of the modeled system and it is formally defined as follows:

Definition 1. *[2] An MDP is a tuple $\mathcal{M} = (S, s_0, Act, \mathbf{P}, L)$, where S is a finite set of states with s_0 the initial state, Act a finite set of actions and $L : S \to 2^{AP}$ maps states to a subset of a given set of atomic propositions AP. The transition probability matrix $\mathbf{P}$ is a function $\mathbf{P} : S \times Act \times S \to [0,1]$. For all states $s \in S$ and actions $\alpha \in Act$, we require that $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0,1\}$. We also require that for each $s \in S$ there is at least one $\alpha \in Act$ with $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$.*

With $Act(s) = \{\alpha \mid \sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1\}$ we specify the set of enabled actions in a state. Fig. 1 introduces a simplified representation of our MDP state space and the enabled actions in each of the shown states. Every state s_i corresponds to the number of VMs that compose the application cluster (e.g. the NoSQL cluster used in [5]) with i representing the cluster's size at some time instant. This illustration of the state space is separated in time sections $(t, t+1, t+2, ...)$ with each one corresponding to a distinct decision step of the cloud provisioning policy. We can thus take into account the evolution of the conditions with time, which is particularly important when a decision policy is coupled with external load prediction[2]. After *remove* and *add* VM actions, the decision maker may be idle for a pre-specified time period (e.g. one decision step) to allow the system to stabilize. In Fig. 1, s_{3-4} at *t+1* and all other states identified with s_{i-j} represent *transient* states, i.e. unstable system states due to a recent change in the number of active VMs. Thus, based on the enabled actions at t, we have three states at $t+1$ including two *stable* states s_2 and s'_3 - if the number of VMs is not changed - and one transient state. States s_3 and s'_3 represent a configuration with 3 VMs, however as the environment evolves, these two states can behave differently to the incoming load (e.g. they may receive different incoming load and may be characterized by different response latency). Also, as we observe, after the s'_3 state, the same pattern is repeated with different time sections and state naming conventions, with s'_3 now being the current state.

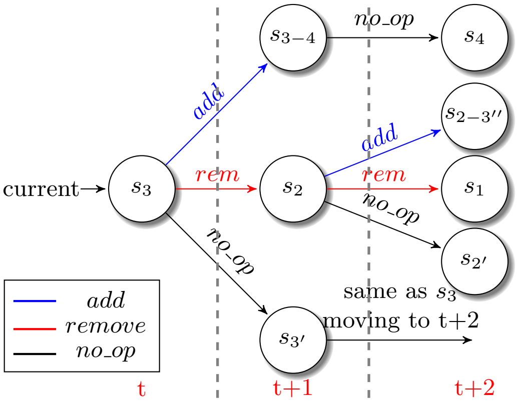

Fig. 1. MDP model overview.

The model's representation in Fig. 1 is further elaborated in Fig. 2 to account for the possible variability in the application's performance for a given external load and cluster size. In Fig. 2, s_i can be any possible *steady* state of the previous model view, where each s_i is in fact represented by n states (shown as $s_i b_m$, $1 \leq m \leq n$). In [5], the most effective decision policies, for a given number of VMs, employ

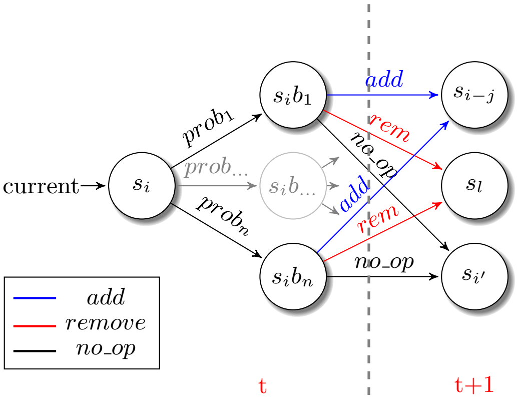

Fig. 2. Detailed MDP model states.

[2] An ARIMA-based predictor of future load can be used in decision policies as suggested by [5].

one state representing the possibility of not meeting the latency threshold and other states representing the possibility to meet it. Transition probabilities are based on the prediction of the future incoming load and the collected logs (cf. Section 3).

Definition 2. *[2] A reward structure for an MDP with state space S and action set Act is a partial function $\mathbf{r} : S \times Act \rightharpoonup \mathbb{R}_{\geq 0}$ assigning a reward to each state and enabled action.*

In our MDPs, a reward is assigned in states based on measurements of the system's latency and throughput, and a user-defined utility function.

3 MDP model instantiation and verification at runtime

A parametric MDP in PRISM's input language is used, which implements the described model representation. We use scripts to automatically generate the model by taking into account (i) system-specific parameters such as the minimum/maximum allowed number of VMs, the allowed numbers of simultaneous VM additions/removals in a single decision step, the lookahead prediction steps, the duration of the transient states, the number of states representing each cluster size, and (ii) system measurements such as the response latency. Our PRISM model's structure is accessible in [3]. Every model is defined as the parallel composition of three PRISM modules:

1. The *decision* module, where the actual elasticity is modeled (add, remove and no_op actions between model states).
2. The *transient* module, which stores measurements for stable states from which an add action is applied, thus reaching a transient state. These measurements are used to compute the rewards for the transient states.
3. The *cluster* module, where a possible state representation as in Fig. 2 is chosen; here the name "cluster" stems from the log measurements clustering in different groups (states) for the same external load and VM cluster size, as explained below and detailed in [5].

Appropriate PRISM formulas are also used to: (i) find the current system measurements, (ii) define the utility function for reward computation and (iii) control the computation of rewards for specific model transitions.

Dynamic MDP instantiation. A new MDP is instantiated in each decision step based on current and logged measurements gathered through periodic monitoring (e.g., every 30 secs). After a preprocessing phase, the measurements are fed into the parametric model to create a model instance for the current and the possible future states.

For the experiments in [5], all model parameters are derived from clustering log measurements for similar past conditions, where the similarity is identified based on the external load, the number of VMs and the response latency. This is an extension of the approach in [6], in order to capture the inherent uncertainty in the external environment. In each elasticity step, log measurements are grouped for a specific number of active VMs by their incoming load λ, and they are then fed into a k-means clusterer, which returns k center points. The k returned centers are mapped to probabilities, which are proportional to the size of their clusters. Finally, the state representation of Fig. 2 is used during the MDP model instantiation, with the computed center points (states) and their respective probabilities.

Model verification and decision making. The elastic decision is based on the maximum expected reward after a specified number of steps. This property is expressed in Probabilistic Computation Tree Logic (PCTL) as follows: $R\{``cumulative\_reward''\}max =?\ [\ F\ (stop)\ ]$. The model checking result is the expected maximum cumulative reward and the set of adversaries which yield this reward, i.e. functions $\delta : S \to Act$ that resolve nondeterminism in the MDP by choosing which action to take in each state.

From the obtained adversaries, we choose the first action of the adversary with the least maximum expected probability for a system measurement threshold violation. The used PCTL properties for this purpose have the form: $Pmax = ?\ [\ F\ (stop)\ \&\ (meas > meas\_threshold)\ \&\ (first\_action = X)]$, where $meas$ is a specific system measurement and X denotes every possible initial action of the processed adversaries. This result represents the preferred elasticity decision.

4 Conclusions and future work

We introduced a parametric MDP model for the control of cloud resource provisioning. This model is the key-component of various elasticity decision policies that have been experimentally evaluated [5] using traces from a real NoSQL database cluster under constantly evolving external load. To the best of our knowledge, this is the first initiative towards integrating model checking in cloud elasticity management and the results in [5] show that our model-driven policies outperform compared to existing alternatives in that they can decrease under-provisioning, while at the same time avoiding over-provisioning.

Latest results show that the presented model checking approach at runtime can be also beneficial for the management of trade-offs between the need to meet performance requirements, while respecting critical constraints in cloud security. To this end, we currently evaluate the effectiveness of a new security-aware elasticity policy on scaling NoSQL databases in a cloud environment [4].

References

1. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. pp. 585–591 (2011)
2. Kwiatkowska, M., Parker, D.: Automated verification and strategy synthesis for probabilistic systems. In: Hung, D.V., Ogawa, M. (eds.) ATVA'13. LNCS, vol. 8172, pp. 5–22. Springer (2013)

3. Naskos, A.: Probabilistic Model Checking at Runtime for the Provisioning of Cloud Resources - Appendix. http://anaskos.webpages.auth.gr/wp-content/uploads/2015/06/parametricMDPmodel.pdf (2015), [Online; accessed 30-April-2015]
4. Naskos, A., Gounaris, A., Mouratidis, H., Katsaros, P.: Security-aware elasticity for nosql databases. In: MEDI (2015)
5. Naskos, A., Stachtiari, E., Gounaris, A., Katsaros, P., Tsoumakos, D., Konstantinou, I., Sioutas, S.: Dependable horizontal scaling based on probabilistic model checking. In: CCGrid. IEEE (2015)
6. Tsoumakos, D., Konstantinou, I., Boumpouka, C., Sioutas, S., Koziris, N.: Automated, elastic resource provisioning for nosql clusters using tiramola. In: CCGrid. pp. 34–41 (2013)