Noname manuscript No.
(will be inserted by the editor)

Cost-aware Horizontal Scaling of NoSQL Databases using Probabilistic Model Checking

Athanasios Naskos · Anastasios Gounaris · Panagiotis Katsaros

Received: date / Accepted: date

Abstract In this work we target horizontal scaling of NoSQL databases, which exhibit highly varying, unpredictable and difficult to model behavior coupled with transient phenomena during VM removals and/or additions. We propose a solution that is cost-aware, systematic, dependable while it accounts for performance unpredictability and volatility. To this end, we model the elasticity as a dynamically instantiated Markov Decision Process (MDP), which can be both solved and verified using probabilistic model checking. Further, we propose a range of complementary decision making policies, which are thoroughly evaluated in workloads from real traces. The evaluation provides strong insights into the trade-offs between performance and cost that our policies can achieve and prove that we can avoid both over- and under-provisioning.

Keywords cloud elasticity · probabilistic model checking · quantitative verification · autonomic computing · PRISM · NoSQL databases

1 Introduction

Cloud computing has arisen as one of the most attractive alternatives for providing computational infrastructures for high-demand applications. The quick prevalence of clouds is fueled by their capacity of achieving economies of scale. One of the main advantages of cloud computing is that it renders the procurement of expensive computing resources unnecessary, thus lifting the burden of high upfront investments in proprietary platforms from system developers and owners. This characteristic is complemented by the capacity for on-demand resource provisioning based on the actual current requirements; this feature is commonly referred to as elasticity, and it is the main focus of this work.

Elasticity is defined as *"the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible"*[25]. Cloud computing extensively leverages virtualization technology. Most commonly, it provides computational resources in the form of virtual machines (VMs). Elasticity may be manifested in different forms. It can refer to runtime modifications of the size, the location or the number of VMs employed, and the combinations of these three basic elasticity types. Common examples of elasticity include the allocation of more memory to a VM (sizing or scaling-up or vertical scaling), moving a VM to a less loaded physical machine (migration) and increasing the number of VMs (scaling out or horizontal scaling) of an application cluster, respectively. Comprehensive surveys on elasticity techniques can be found at [36] and [13].

In this work we target the third type of elasticity, horizontal scaling, in a specific setting, namely NoSQL databases. This setting has the following two characteristics that differentiate it from simple cloud-hosted applications: (i) Increasing or decreasing the number of VMs is a key element in adapting to dynamically changing volumes of user requests. However, the behavior of the system is unpredictable, significantly varying and

Athanasios Naskos
Aristotle University of Thessaloniki, Greece
E-mail: anaskos@csd.auth.gr

Anastasios Gounaris
Aristotle University of Thessaloniki, Greece
E-mail: gounaria@csd.auth.gr

Panagiotis Katsaros
Aristotle University of Thessaloniki, Greece
E-mail: katsaros@csd.auth.gr

unamenable to analytical modeling due to the complexity of the underlying mechanisms serving user requests within a modern database; and (ii) there are significant transient periods, during which the effects of an horizontal action are not apparent.

A solution to horizontal scaling decision making for NoSQL databases should have the following characteristics:

1. *To be systematic and dependable.* We meet this requirement through a proposal that is based on a solid theoretical background, namely analysis of Markov Decision Processes (MDP). Such an approach is opposed to more ad-hoc solutions that are based on thresholds (or rules), e.g., [6, 7, 23, 39, 33, 3, 12, 46, 21, 15, 10, 31, 5], which are known to be difficult to set appropriately [17]. Further, to attain dependability, we resort to continuous verification of the system MDP model, using probabilistic model checking [20].
2. *To account for performance unpredictability and volatility.* Our solution meets this requirement through runtime model instantiation taking into account the latest log measurements. As such, it does not rely on offline modeling or other approaches that silently assume that the system is fixed, e.g., as in [44, 2, 28, 1, 45].
3. *To consider multiple objectives.* Horizontal scaling is an inherently multi-objective problem. More resources are required during periods of peak load to keep performance within user-defined specifications. However, additional resources should be employed in a judicious manner, because they incur economic cost, either implicitly (e.g., higher energy bills in private clouds) or explicitly (e.g., more VM hours in public clouds offering resources using an hourly-based charging model). Our proposals explicitly consider performance and economic cost when using VMs of a public cloud provider. More specifically, we aim to minimize the monetary cost keeping the latency of responses to users below a threshold; this issue has not been investigated for NoSQL databases to date.

In summary, the contribution of this work is the first proposal to date for monetary cost-aware horizontal scaling tailored to NoSQL databases. As mentioned above, a distinctive feature of our proposal is that it uses probabilistic model checking at runtime as the main decision mechanism. Further, it accounts for unpredictable and volatile system behavior.

An additional strong feature of our approach is that it follows a decoupled design model, where the underlying system model can support various decision making policies. Furthermore, we provide full implementation details covering aspects such as incorporating mechanism to predict future external load, to smooth the incoming load requests.

Our work builds on our previous work in [37, 38], where we have shown the applicability of probabilistic model checking in elasticity decision making. We have significantly extended our previous work in that we explicitly consider monetary cost (accounting also for the commonly used hourly-based charging model) and we devise novel elasticity decision making policies, which are proven to significantly outperform the results in [37, 38]. In summary, our evaluation is based on real traces and the results show that we can strike a configurable balance between under-provisioning and over-provisioning. In a scenario, where using no extra machines leads to performnance violations in more than 50% of the time steps, these violations are decreased to less than 1%, while using no more than half of the maximum number of machines on average. More importantly, the different decision policies that we propose are complementary (i.e., they do not dominate each other) and can achieve different trade-offs between performance and cost at a fine level of granularity.

The remainder of the article is structured as follows. In the next section, we discuss related work. In Section 3, we present the main aspects of interest regarding the elastic behavior of NoSQL databases. Our modeling methodology is detailed in Section 4. The novel contribution of our work, i.e., cost-aware policies, is presented in Section 5. We evaluate our proposals in Section 6. The conclusions are in Section 7.

2 Related Work

In a nutshell, our proposal differs from other proposals for horizontal scaling both a) in terms of its decision approach, that is to employ model checking for runtime elasticity decisions and not to rely on offline modeling; and b) in that it proposes bi-objective cost-aware horizontal scaling policies tailored to NoSQL, which exhibit unpredictable and volatile behavior even for the same external conditions, and each elasticity action is followed by a significant transient period.

We split related work in three parts. First, we discuss elasticity proposals that are cost-aware but do not consider the specificalities of NoSQL databases. Then, we move our attention to proposals for NoSQL databases, which, however do not consider monetary cost. Third, we discuss solutions that also employ MDPs and highlight the differences from our.

2.1 Cost-aware Elasticity

Monetary costs have been considered in several elasticity proposals. In the simplest case, the horizontal scaling actions are defined as rules that are triggered upon the violation of a threshold and the monetary cost does not consider pragmatic issues, such as hourly-based VM provision. Examples that fall in that category are the proposals in [11, 46, 22]. In [40], an approach to generate such rules dynamically is presented.

Another class of related proposals suffers from the fact that are based on offline model training, as explained in the introduction. An example is [19], which does not consider pragmatic charging models either. Another example is [4], which relies on a PID controller.

Proposals, such as [11, 46, 8, 32], consider a different version of the problem, where there is a budget limit. We do not put any constraint on the monetary cost, but our approach is flexible in that it devises policies that either minimize that cost or trade it for performance.

Also, the techniques in [11, 46, 8, 4, 43, 27] assume a different application setting, such as multi-tier web applications, where it makes sense to consider application reconfiguration along with horizontal scaling. We consider only the latter targeting NoSQL databases. [46] examines vertical scaling actions as well. Similarly to our work, in [43], look-ahead optimization of a weighted multi-objective function, is used to find the optimal scaling and/or the optimal application reconfiguration in terms of cost to performance ratio. In [27], a reactive policy is proposed, where an elastic action is triggered every time a request to a web application queue exceeds a threshold. Their policy takes into consideration the hourly-charging, as it evaluates the need for resources for the next hour, utilizing an objective function, which is optimized using exhaustive search. The objective function includes both VM cost and SLA violations (i.e. response latency violations).

There are also proposals that directly target the cost of the deployment to handle the elasticity [47]. The latter, utilizes both horizontal and vertical scaling to proactively scale the VMs cluster and/or apply self-healing scaling (i.e. provisioning redundant occupied resources to VMs that are close to be saturated) as a reactive policy. Linear regression in combination with a greedy heuristic algorithm is used to solve an optimization problem which targets to minimize the deployment cost, occupying the least number of resources, while trying to satisfy more user requests. [9] is framed in the context of application-level resource provisioning, where reactive and proactive heuristic approaches are proposed to solve an optimization problem, which tries to find the best trade-off between the allocation cost and the SLA violations (i.e. response latency).

Finally, [18] approaches cost-aware elasticity from the data center owner point of view, and as such considers different types of cost, including hardware cost, license and labour cost, and penalties due to SLA violations. On the contrary, we view the problem from the cloud application point of view.

2.2 Elasticity for NoSQL databases

Similar to our work, [45] considers the horizontal scaling of a NoSQL cluster, where the system is described using a MDP. They propose an indirect solution to the MDP (i.e. Bellman equation), which uses a Q-Learning approach. This work has been significantly extended by our previous work in [37], which, in turn, is extended hereby. The authors in [1] propose a feedforward controller for key-value stores, which monitors the workload and uses a logistic regression model to predict whether the workload will cause SLA violations and react accordingly. This controller is combined with a feedback controller, which monitors the performance and reacts based on the amount of deviation from the desired performance specified in the Service Level Objective (SLO). Nevertheless, it requires offline modeling.

There are also works that combine the horizontal scaling with other forms of system modifications, such as system reconfiguration actions [16, 14, 10]. In [16], neural networks are used to estimate the throughput and response time of the in-memory data store (i.e. Infinispan) and then, a controller solves a constraint optimization problem to determine an optimal resource configuration in terms of number of VMs and data replication degree. [14] uses a rule based policy to define if a variant of a bin-packing problem should be solved, to define if a horizontal scaling to the number of VMs or a scaling in the maximum number of data partitions per node is needed. [10] targets Cassandra and propose an approach that utilizes horizontal scaling in combination with cache size dynamic re-configuration. Finally, in [42] database migration on multiple cloud providers is applied, to obtain a trade-off between the deployment cost and the SLO violations. In all these works, the horizontal scaling part is less sophisticated than ours, however, it is interesting to combine horizontal scaling with additional elasticity actions in the future.

2.3 MDP-based Decision Making

There are also works that utilize MDP modeling to guide the decision making [24, 35, 34, 45]. In these

works MDP is only used for the formation and the solution on an optimization problem. We also utilize MDP for optimization, however we additionally apply probabilistic model checking for the analysis of the model, offering a more dependable decision making approach providing probabilistic guarantees. In [24, 45], authors propose an indirect MDP solving approach, using Bellman equations. [24] propose an approximation to the optimal Bellman solution, reducing the state space of the problem. Moreno et al. in [34], utilize PRISM and PCTL to formulate and solve the MDP model similarly to our approach. In a more recent work Moreno et al. [35], propose a more efficient MDP model solving approach based on Alloy models [26], however their approach is only appropriate for optimization problem solving and does not allow for probabilistic model analysis.

3 NoSQL elastic behavior

NoSQL databases are designed to be distributed across multiple nodes.[1] Deploying a NoSQL database across multiple VMs leads to some form of performance unpredictability. In this section, we provide experimental evidence regarding query response variations (i) when horizontal elasticity takes place, and (ii) when the cluster setup and the external load remain stable.

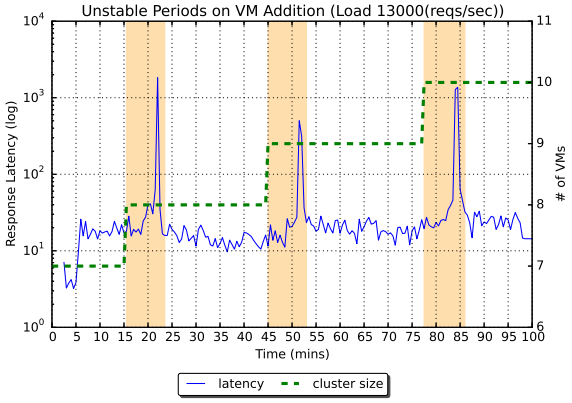


Fig. 1 Unstable periods after a scale-out elastic action

Figure 1 refers to a scenario where a Cassandra database serves requests at a rate of 13000 requests/sec according to the YCSB (Yahoo! Cloud Serving Benchmark) and the number of VMs on which the database is deployed is progressively increased from 7 up to 10 VMs (green dotted line). In the figure, the blue solid line presents the response latency. We can see that each node addition is followed by an unstable period, i.e., a period with high response latency, which is highlighted in the diagram. The duration of such a period is approximately 10 mins. During this period, the data allocated to the new VM are transferred to its memory from their previous hosts in a lazy manner (i.e. data is transferred upon request).

[1] In this work, the terms node and VM are used interchangeably.

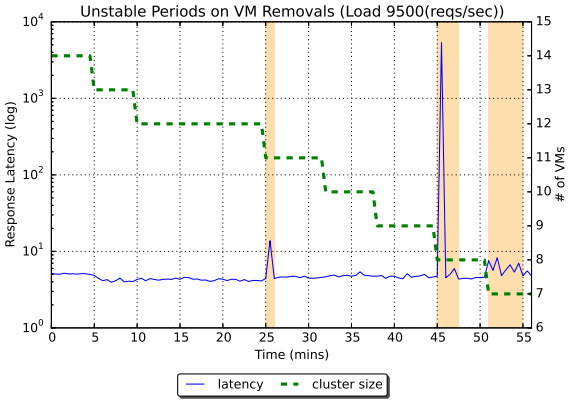


Fig. 2 Unstable periods after a scale-in elastic actions

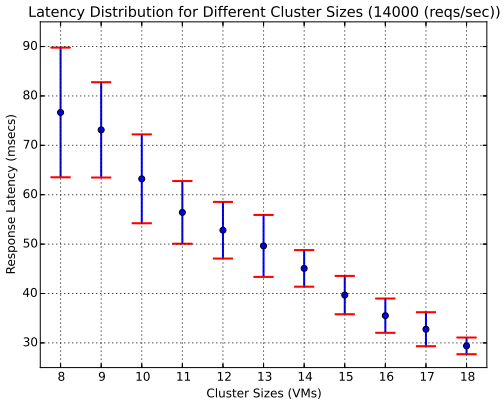


Fig. 3 Latency Distribution for Different Cluster Sizes (14000 (reqs/sec))

Considering the node removals, there are two options in the NoSQL databases, (i) the instant removal and (ii) the graceful removal. In the first case, removing a VM does not lead to data loss due to replication and the node can be removed as soon as it serves its pending requests; then a background process kicks-off to create a new replica for the data previously hosted on the removed VM. In the second case, the node stops serving requests and distributes its tuples to the rest of the server nodes; when this processes finishes, the node is removed from the cluster. Both options cause unstable periods mostly due to the network transfer overhead, however the second one affects more the performance of the system, as the self-healing of the first case can be applied in a lazy manner. This is the rea-

son that the second case is recommended to be applied only in cluster idle hours. In Figure 2, we present the first case of node removal in the same Cassandra cluster, where the cluster size is progressively reduced from 14 to 7 server nodes. The external load remains 9500 reqs/sec throughout the experiment. As can be observed, when the cluster size to handle the external load is large enough (i.e. 9-14 server nodes), there are almost no unstable periods. As the cluster size reduces, short unstable periods are observed with duration less than 4 mins. These unstable periods of the cluster need to be taken into consideration in the elasticity decision making process, as they can greatly degrade the QoS offered to the clients.

Another distinctive behavior of cloud-hosted NoSQL databases is that the latency of responses varies significantly, even when the number of VMs and the external load remain stable. Figure 3 shows the average and standard deviation values of the latency of responses for different cluster sizes. The standard deviation of the response latency is very high, especially for smaller cluster sizes as the cluster strives to serve an amount of requests with not enough VMs. As can be inferred from the figure, it is common larger clusters to perform better than smaller ones. Taking into account the performance variability during elasticity actions is essential for developing robust decision policies.

4 Probabilistic Model Checking for Elasticity

Probabilistic model checking is a formal verification technique for the modeling and analysis of stochastic systems [30]. In our work, probabilistic models are used in the decision making process, to describe, drive and analyze cloud resource elasticity. By utilizing probabilistic models, we are able to capture the uncertain behavior of systems elasticity. In order to additionally capture non-determinism, we resort to Markov Decision Process (MDP) models, which form the basis of our approach. On top of our MDP models, we build policies for elasticity decisions with the help of the PRISM probabilistic model checker [30].

Problem Description: More formally, we target a bi-objective function as follows: minimize *cost* through tuning the number of VMs, provided that the latency *lat* is kept below a threshold lat_{thres}.

Given that, when user requests are answered in a bounded time, then the throughput can closely follow the incoming load, the objective function ensures that the system copes with the volatile load in a cost-efficient manner.

4.1 MDP basics

MDPs provide a mathematical framework for modeling decision making in situations, where outcomes are partly random and partly under the control of a decision maker [41].

An MDP is a tuple

$$M = (S, s_{init}, Act, P_{sas'}, L, R),\ where$$

- $S = \{s_0, ..., s_n\}$ is a finite set of states;
- s_{init} is the initial state;
- $Act = \{a_0, ..., a_m\}$ is a finite set of actions;
- $P_{sas'} = Pr\{s^{t+1} = s' | s^t = s, a \in Act\}$ is a transition probability from state s at step t to state s' at the next step due to action a;
- L is a finite set of state labels; and
- $R = (r_s, r_a)$ is a pair of reward functions, with $r_s : S \rightarrow \mathbb{R}_{\geq 0}$ assigning state rewards and $r_a : S \times Act \rightarrow \mathbb{R}_{\geq 0}$ assigning rewards to all state-action pairs.

The resolution of non-determinism is called a *strategy* or *policy* or *adversary*, and is defined as a function $\sigma : S \rightarrow Act$, which maps states to concrete actions. However, in this work, we use the term *policy* not in this sense but in the sense of choosing among multiple candidate adversaries (or strategies), as explained in Section 5.

4.2 High-level model description

In this work we propose a modeling approach, which allows us to map the elasticity problem to a model checking one, applying classical model verifications techniques to solve it. Figure 4 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 (i.e., a NoSQL cluster) with i being equal to the corresponding number of VMs. We also need to take into consideration the evolution of the environment. In our proposal, we periodically monitor the evolution of the system's conditions. The period of the activation of the decision making mechanism, called *decision step*, is explicitly captured by the model. To cover the evolution of the environment through the time, the state space is conceptually separated in time sections $(t, t+1, t+2, ...)$, where t is the activation period and each section corresponds to a distinct *decision step*. Overall, each state refers to a unique combination of cluster size and decision step, but the opposite does not hold (i.e., each size-step combination may be covered by multiple states), as will be explained shortly. An MDP allows to capture both the non-deterministic and the probabilistic

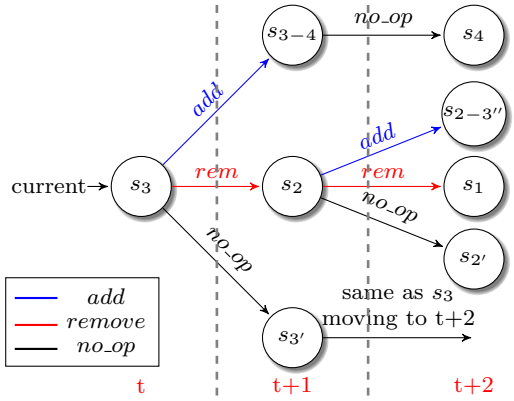


Fig. 4 MDP model overview.

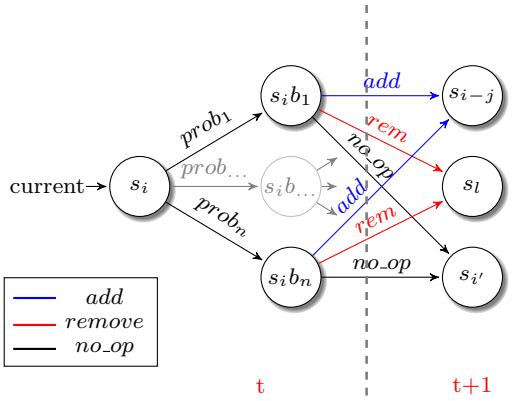


Fig. 5 Detailed MDP model states.

aspects of the modeled system. In the elasticity domain, we are interested in making decisions over a set of possible options that give rise to non-determinism, i.e, adding, removing or maintaining the number of VMs, according to the system's behavior monitored online. Each such option, is a different action in the model. Actions connect states only from consecutive steps.

After *remove* and *add* VM actions, the decision maker may be idle for a pre-specified time period (i.e. time period to allow the system to stabilize (see Section 3)). In Figure 4, 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. Transient states need not to be explicitly captured if the decision step is shorter than the transient period; e.g., in the figure we assume that the decision step is longer of the transient period after VM removals. Overall, 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/or 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.

4.2.1 Extended model states to capture variability

In our representation, every state, is labeled by some measurements specific to NoSQL databases. The most relevant to our problem is the envisaged latency corresponding to a specific state at a given future step, which is derived through log measurements. The main problem encountered is that, as shown in Figure 3, latency suffers from high variability. To work around this, we split each state into smaller ones with more predictable behavior as shown in Figure 5. In the figure, s_i can be any possible stable state of the previous model high-level view, where each s_i is in fact represented by n states (shown as s_ib_m, $1 \leq m \leq n$, where b stands for behavior). Transition probabilities are based on the collected logs.

4.2.2 Rewards and Utility Functions

Our decision making proposal uses state rewards in the model verification process, which are computed based on a user-defined utility function. To comply with the rest of the model, utility functions can be functions of the number of active VMs, the response latency, the deployment cost or any other parameters considered in our model. These utility functions are used to derive the state rewards in each model instantiation and their optimization (i.e. maximization or minimization) will guide the elasticity decision making. Numerous utility functions can be used in elasticity scenarios. In the next section, we will present utility functions that penalize both under- and over-provisioning taking into consideration both performance and cost issues.

4.3 Model instantiation details

At each decision step, a MDP model (illustrated in Figures 4 and 5) is constructed automatically based on a template. Thus the instantiated model reflects the current system and environment conditions. Here, we briefly outline a series of practical considerations.

We construct the states in Figure 5 in a principled manner. More specifically, for each time step, we predict the expected external load. For that external load (allowing for some variations), we extract the past latency log measurements, for a specific of number of VMs and we perform a guided clustering procedure. One group

refers to the latencies that exceed the threshold in the problem definition, whereas the other ones are clustered using the k-means algorithm. Each latency group is then mapped to a distinct state. In this work, n is set to 3. More details are provided in [37].

On the lack of past latency measurements for a specific state, we generate artificial measurements using interpolation or extrapolation techniques. The generated measurements might not be close to the actual state measurements, leading to inferior decisions, however once our system gets logs from the missing states, in the next decision step the optimal decision will be selected.

As explained above, the model states are instantiated dynamically in each decision step according to the existing log measurements for the current and the future external load λ. To predict the latter, any prediction mechanism can be coupled with our approach.

Finally, to tackle sudden and temporal peaks of the system load, which trigger suboptimal elasticity actions, we (optionally) utilize smoothing techniques to the incoming load. The two main approaches that proved to be effective in our case studies are: (i) to use the average value of the incoming load or (ii) to use an exponential weighted moving average (EWMA), assigning higher significance to the more recent measurements, given a sliding window.

4.4 Generalization to heterogeneous clusters

In principle, our modeling approach can be naturally extended to capture heterogeneous clusters. The main difference is that for each high level state s_i, we construct as many states as the possible combinations of VM instances of different types that amount to i. For a cluster of size i comprising two types of VMs, this mapping yields i different states. Obviously, this technique is applicable to settings with a small number of different VM types due to scalability issues.

4.5 Solving MDP models

There are several ways to solve MDPs. We distinguish between indirect (based on reinforcement learning) and direct methods (based on dynamic programming). Indirect methods are exemplified by the approach in [45], which relies on online training and convergence of action-value functions, which, in turn allows to attain optimal adversaries through greedy actions and a Q-learning-based reinforcement learning approach. The direct methods analyze MDPs per se. In our approach, we adopt direct solutions and we use the PRISM tool to this end, because we do not only solve the model online but we also perform online property verification.

The verification of the MDP models is based on Probabilistic Computation Tree Logic (PCTL) properties. PCTL allows for probabilistic quantification of described properties. It is utilized to query the model checker for the validity, the value or the existence of a specific condition, considering probabilities, rewards or combination of them, in the model. PCTL is an expressive logic, that allows the formation of complex properties to express more complex queries [30].

The elastic decision is based on the optimization of the expected reward after a specified number of steps (max_steps). This property is expressed in PCTL as follows:

$$R\{cum\_reward\}[max/min] =? \ [\ F\ (max\_steps)\ ] \quad (1)$$

The model checking result, which is the results of the PCTL property verification, is the expected optimized (maximum or minimum) cumulative reward and the set of strategies which yield this reward, i.e. functions $\delta : S \rightarrow Act$ that resolve non-determinism in the MDP by choosing which action to take in each case.

Typically, more than one strategies belong to the result set. To derive the elastic action to be enforced, we may utilize an additional PCTL query that performs quantitative analysis. A typical choice is to select the first action of the strategy with the least maximum expected probability for latency violation. The used PCTL property for this purpose has the form:

$$Pmax =? \ [\ F\ (max\_steps)\ \&\ (lat > lat_{thres})] \quad (2)$$

We verify this PCTL property for every candidate initial elastic action. This approach forms the basis for our solutions, which are elaborated in the next section.

5 Policies for Elasticity Decisions

In this section, we present the proposed elasticity decision making policies, which are summarized in Figure 6. The purpose of the policies is to specify how to choose an adversary among multiple candidates. After an adversary has been chosen, then, only its first action is enforced. The adversaries are reconsidered in each decision step.

In the figure, each rounded rectangle (in blue) corresponds to a different policy. The normal rectangles (in red) correspond to the evaluation of the properties in the Eq. (1) and Eq. (2). The circles (in red) correspond to the re-evaluation of the previous properties after having relaxed the strict optimality criteria, thus allowing for results with non-optimal rewards and

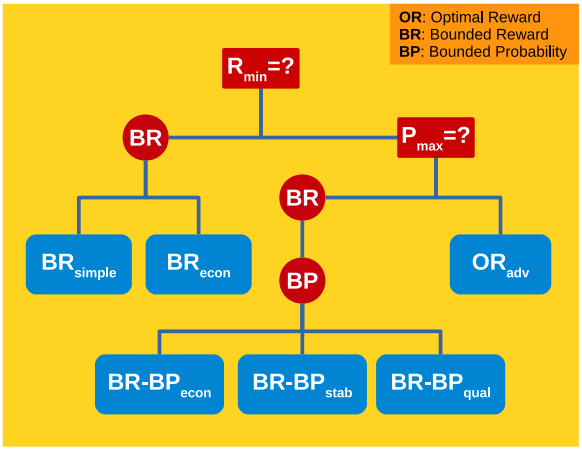


Fig. 6 Proposed Decision Policies

probabilities, respectively. The deviation from the optimal value is bounded. The rationale of allowing for sub-optimal policies is to consider alternative strategies that may strike additional trade-offs between cost and latency threshold violations.

In this work, we propose five new decision policies in addition to the best-performing one from [37], namely OR_{adv}[2]. These policies are discussed in turn below and assume that the utility functions employed during reward computation are to be minimized (as explained later):

* *Optimal Reward Advanced (OR_{adv})*: The optimal elastic action is selected based on the lowest cumulative reward, with the least maximum probability of latency violations. If there are more than one candidate actions, the action that removes the most VMs is selected. On the absence of a remove action, the one that applies the least change in the number of VMs is preferred.

1. *Bounded Reward Simple (BR_{simple})*: The goal of this policy is to reduce the deployment cost. 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. If there are no such actions, then the optimal elastic action in terms of Eq. (1) is applied. This policy is activated only if the optimal elastic action (i.e., the first action of the optimal strategy) is an addition one.
2. *Bounded Reward Economy (BR_{econ})*: Similarly to the previous one, this policy is activated only if the optimal action is an addition one and 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. If there is no such action, *no operation* is selected (if available).
3. *Bounded Reward - Bounded Probability Economy ($BR-BP_{econ}$)*: This policy is also activated if the optimal action is an addition and 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 (see Eq. (2)) falls below a given threshold. If there is no *remove* action returned, then the *no operation* action is selected. If *no operation* action is candidate, the action that adds the least number of VMs is selected.
4. *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. Its rationale is to proactively add VMs, which are likely to be needed anyway in the future thus avoiding latency violations.
5. *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. It firstly chooses the *no operation* action, if available in the candidate set (i.e., the strategies with reward up to $x\%$ higher than the optimal. Then it chooses the action that is going to add the smallest number of VMs and finally, as its last option, it chooses the action, which is going to remove the lowest number of VMs.

5.1 VM charging model awareness

Typical charging models of cloud VMs are per hour; an example is Amazon EC2 pricing scheme for on-demand VMs.[3] As such, it makes no sense to remove a VM early after it has been used for another hour, since it is paid for the complete hour. Our proposed elasticity decision policies are enhanced with VM running time awareness to better handle the VM removal process. According to this enhancement, we only remove VMs that are close to the completion of a full hour of execution, as we use an hourly based charging policy for the provisioning of the VMs. More specifically, a user-defined parameter within

[2] This policy was originally proposed in our previous work [37] named as *ADVANCED*.

[3] `https://aws.amazon.com/ec2/pricing/`

each policy defines the running time of a VM before it becomes eligible for removal. Our default choice is 10 minutes before the completion of a full hour. If there are more VMs eligible for removal (i.e. having running time close to an hour) than the decided number of VMs decided to be removed, the VMs with running time closer to a full hour are selected. Other time units apart from hours can be supported in a straightforward manner.

5.2 Utility Function Types

The presented decision policies are based on the minimization of a user-defined utility function, which reflect the specified goal. In our case the goal is to minimize the deployment cost, while keeping the response latency of the system below a threshold. Given that latency violations are due to under-provisioning, our utility functions penalize the fact of activating fewer VMs than those needed. Although the decision policies above are orthogonal to the exact utility functions used to compute the rewards, their exact type heavily impacts on the behavior and efficiency of the policies; this is also verified by our experiments in the next section. In this section, we present a specific set of utility functions, however our models are extensible and can associate additional variables (e.g., throughput) and utility functions.

More specifically, we have investigated three families of utility functions. The first one is termed as *NWI*, which stands for Not-Weighted Implicit cost consideration, and assigns a reward to a state $s_i b_m$ according to the formula below:

$$NWI(s_i b_m) = \begin{cases} 1 - 1/i & \text{if } lat(s_i b_m) \leq lat_{thres} \\ 2 & \text{if } lat(s_i b_m) > lat_{thres} \end{cases} \quad (3)$$

In NWI, the deployment cost is indirectly considered through the number of utilized VMs i. $lat(s_i b_m)$ represents the expected latency for the state $s_i b_m$.

NWI is applicable to cases of homogeneous resources only. The following two utility functions address this limitation, as they directly take into consideration the cost of every distinct VM type rather than just the number of occupied VMs.

The *NWE* utility function, which stands for Not-Weighted Explicit cost consideration, has a similar form with the *NWI* with the difference that the number of VMs is replaced by the total cost of the deployment.

$$NWE(s_i b_m) = \begin{cases} 2 - 2/(1 + \tilde{cost}(i)) & \text{if } lat(s_i b_m) \leq lat_{thres} \\ 20 & \text{if } lat(s_i b_m) > lat_{thres} \end{cases} \quad (4)$$

$\tilde{cost}(i)$ is the normalized cost of a cluster with i VMs. More specifically, assuming that we know the minimum and maximum cost of VM instance types that can be employed, we normalize the deployment cost ($\tilde{cost}$) to the range $[0, 1]$. To fine-tune the utility function, we have examined a series of different punishment values (presented in Figure 13). Setting the punishment value too high or too low has a negative impact on the deployment cost or the response latency of the system, respectively. A punishment value of 20 as employed by Eq. (4) seems to achieve a more acceptable trade-off.

The third utility function is *WEab*, which stands for Weighted Explicit cost consideration. Here, a weighting scheme is used to provide more configurable trade-offs:

$$WEab(s_i b_m) = \begin{cases} a \cdot \tilde{cost}(i) + b \cdot lat(\tilde{s_i b_m}) & \text{if } lat(s_i b_m) \leq lat_{thres} \\ 2 & \text{if } lat(s_i b_m) > lat_{thres}. \end{cases} \quad (5)$$

a and b are user defined weights with $a + b = 1$ and $lat(\tilde{s_i b_m})$ is the normalized response latency. As there is no upper bound for the latency, we use z-score normalization; then we transform the $[-1, 1]$ range into $[0, 1]$, while values less than -1 (resp. greater than 1) are mapped to 0 (resp. 1).

6 Evaluation of Decision Policies

In this section we compare the efficiency of the decision policies enabled by our approach, in combination with the proposed utility functions. As a baseline, we consider the behavior of OR_{adv}, which is shown to outperform other policies proposed by third parties, including the rule-based one from Amazon's EC2 [37].

6.1 Experimental Setup

We collected real logs from an Apache Cassandra NoSQL cluster, deployed in okeanos IaaS infrastructure [29]. Then, we ran emulated experiments based on those logs, to allow for a completely fair comparison between the various techniques. In order to collect real data from the Cassandra cluster, we conducted log measurement experiments using the YCSB benchmark. For our Cassandra NoSQL cluster, we have used 4 client VMs as load generators with 2 VCPUs and 4GB of RAM and 5GB storage each, and from 8 to 18 Cassandra server VMs with 2 VCPUs, 2GB of RAM and 20GB storage each. The Cassandra version was v2.0.9 with 256 virtual nodes per host and a replication factor set to 1. A (heavily modified) version of YCSB-0.1.4 ran on every client VM to produce the load; the modifications were made to support database metrics reporting on Ganglia.

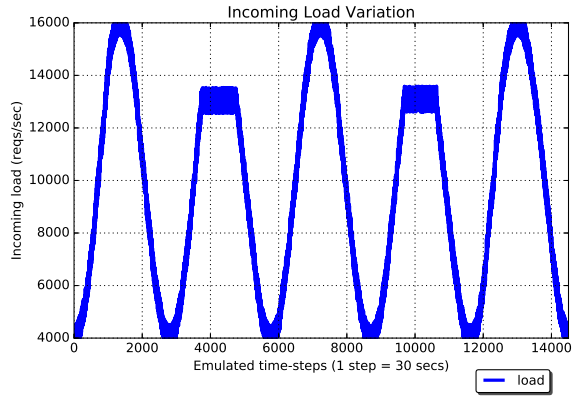


Fig. 7 Incoming Load Variation

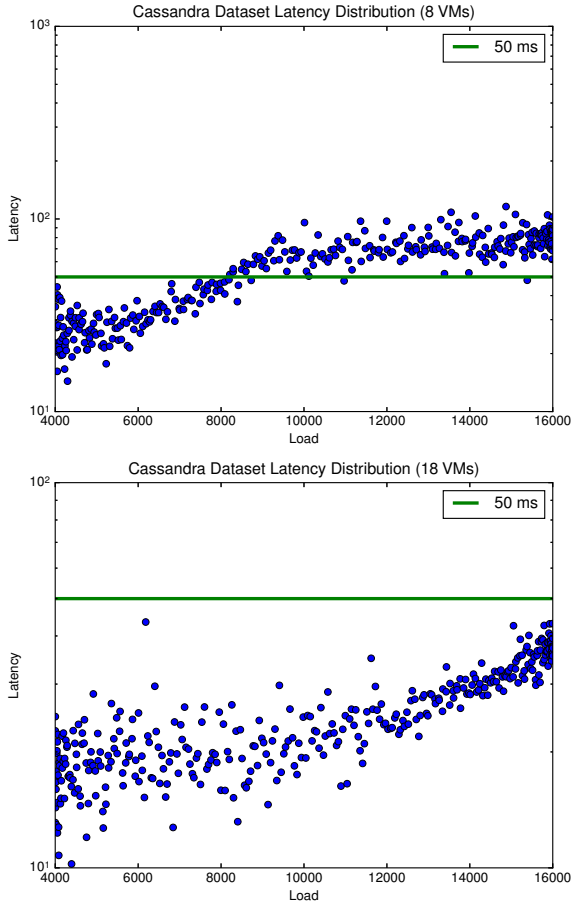


Fig. 8 Latency Distribution for minimum (top) and maximum (bottm) cluster size for Cassandra dataset

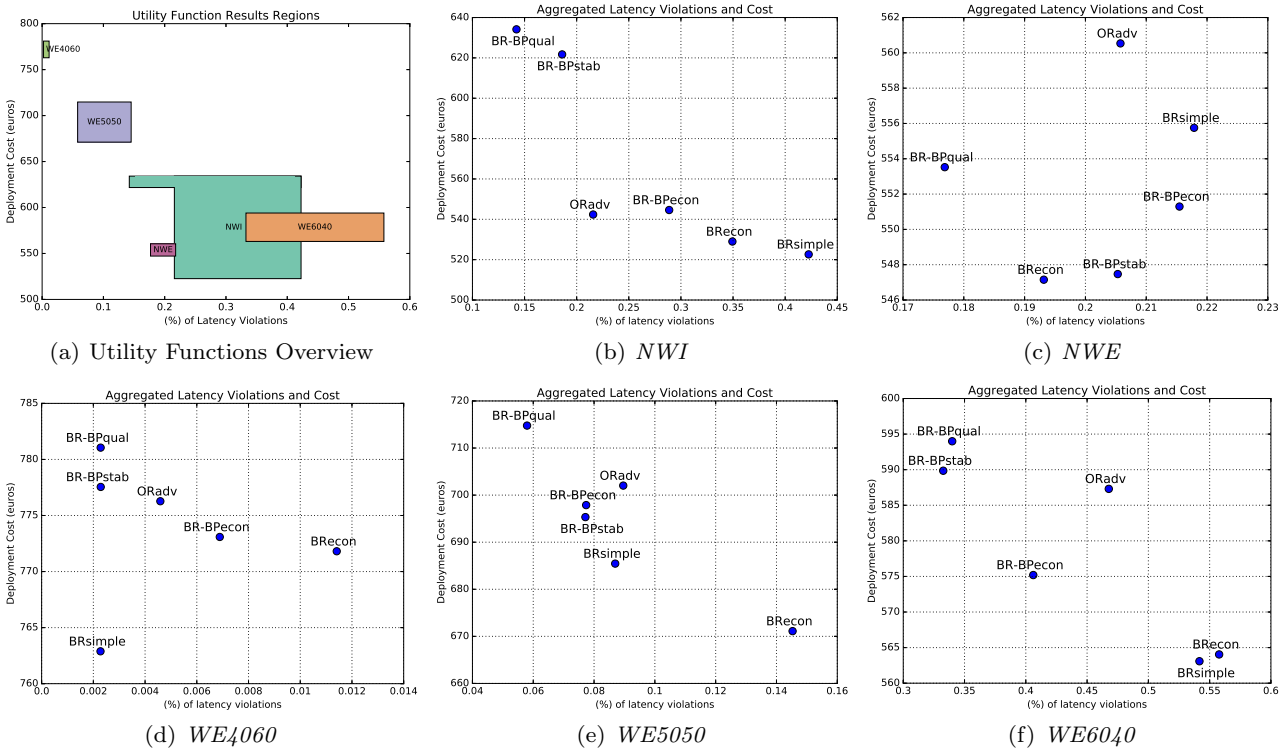
The collected measurements are used firstly, to populate the initial logs of each policy, and secondly, to emulate a real situation. Through emulation, we managed to fairly test each policy on an equal basis, which could not be done if each policy ran separately in a real cluster. In our emulation, a time unit corresponds to the measurement collection period, which was set to 30 secs. We allow an elasticity action to take place every 10 time units, to emulate a system that may modify the VMs every 5 mins (or 10 mins is cases of add action, to allow the system to stabilize). As the emulated load is generated based on the logs, which also act as training set, we consider that the system is well trained.

Figure 7 depicts the load applied during elasticity experiments. It is a 5 period sinusoidal workload with 2 plateau periods. The sinusoidal load varies from 4000req/sec to 16000req/sec and the 2 plateau periods correspond to stable load at 13000req/sec for 1000 time units each. This load is further randomly perturbed by a factor of up to ±500req/sec. To tackle load fluctuations, in all the presented experiments, EWMA is utilized. Further, an emulated prediction mechanism is used to predict the incoming load future values, where the uncertainty is increased as we move further into the future (i.e. $uncertainty = future\_step * (\pm 100 reqs/sec)$, where $future\_step \in [0, max\_steps]$). Finally, in every scale-out action, up to 3 VMs can be added, while during scale-in, up to 2 VMs are allowed to be removed in a single step.

The latency threshold in the utility functions is set to 50ms. Figure 8 presents the latency distribution in two characteristic states of the collected dataset, the minimum number of VMs (top) and the maximum number of VMs (bottom), where the solid line shows the latency threshold. For the minimum number of VMs (8), the system can handle load up to about 8000 req/sec. For the maximum number of active VMs (18), the system can handle the full amount of the incoming load.

For all the sub-optimal policies the acceptance percentage of the sub-optimal reward (BR) is set to +0.5% of the optimal reward. The bound for the probability (BP) for the corresponding policies is set to 5%. For the *WEab* utility function, we present three combinations of weights: (i) a=40%, b=60% (*WE4060*), (ii) a=50%, b=50% (*WE5050*), and (iii) a=60%, b=40% (*WE6040*).

Finally, for the hourly cost of the VMs, we consider that the 8 default VMs are provided through a private cloud infrastructure, hence the cost of these VMs is set to 0. For the rest 10 VMs, we consider that VMs with equivalent features, hence the same performance, are selected from a public cloud provider and more specifically, we have selected the *c3.4xlarge* instance of Amazon EC2, with hourly cost set to 0.956 euros. The actual running time of each experiment in real world amounts is 131 hours, hence the maximum cost of an experiment (i.e. using the maximum number of VMs (i.e. 18) throughout the experiment) is 1252.36 euros (i.e. 131 hours × 0.956 euros/hour × 10 charged VMs). The minimum cost is 0, as only the private cluster's VMs can be used. The maximum percentage of latency vi-



(a) Utility Functions Overview (b) *NWI* (c) *NWE*

(d) *WE4060* (e) *WE5050* (f) *WE6040*

Fig. 9 Aggregated Latency Violations and Deployment Cost for all the utility functions.

olations (i.e. using only the private cluster's VMs) for the given incoming load, is 57.9 %.

6.2 Main Experimental Results

Figure 9, presents the experimental results in the form of the latency violations and monetary costs trade-offs. Figure 9(a) depicts the conceptual regions covered by the applications of all the proposed decision policies for each utility function. The remainder figures zoom in each such region; for visibility reasons, the axis scaling in each figure is different, thus they need to be examined in correlation to Figure 9(a).

The key remarks are summarized as follows:

1. The techniques manage to achieve elasticity actions that avoid both under- and over-provisioning. When no additional machines are employed, the percentage of violations is 57.9%. As can be seen from the figures, the policies manage to drop this number to well below 1%. This is achieved employing additional VMs judiciously. If, at all steps, the full capacity of the cluster is exploited (i.e., 18 VMs), the monetary cost would be 1252.36 euros, whereas our solutions correspond to cost as low as 530 euros (i.e. 58% reduction from the maximum).
2. The behavior of the techniques is utility function-dependent. This is evident by the different conceptual regions (in terms of both boundaries and size) covered by each utility function in Figure 9(a) and the different patterns exhibited by the policies in Figures 9(b)-(f).
3. In general, no decision policy and no utility function dominate. However, OR_{adv} initially proposed in [37] is dominated by the new policies proposed in this work. Also, WE with factor $a > 0.5$ is dominated by NWI/NWE regardless of the exact decision policy.
4. Following on the previous point, our solutions can yield a wide range of trade-offs at two levels of granularity. At a higher level of granularity, the range of trade-offs can be configured through the selection of the appropriate utility function. If latency violations of frequency at the orders of 0.01% are tolerated, *WE4060* should be chosen. If the tolerable frequency is approximately 0.1%, then *WE4060* should be chosen. If the tolerable frequency is even higher, NWI/NWE become the preferable utility functions. At a finer-level of granularity, the desired trade-off between cost and latency violations can be selected through the selection of the exact decision policy.
5. The $BR-BP_{qual}$ policy achieves the least percentage of latency violations with the maximum deployment cost, as expected, in most of the cases. An exception is depicted in Figure 9(c) for the *NWE*

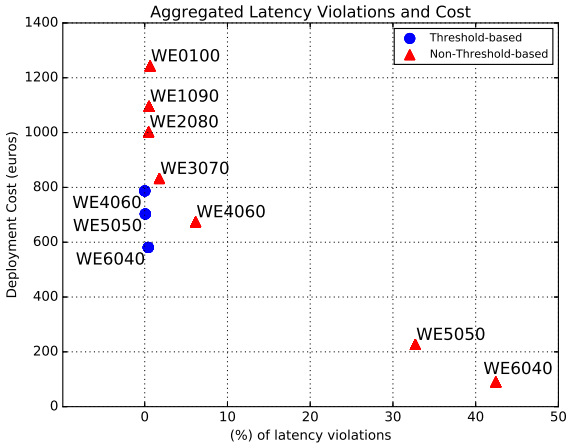


Fig. 10 Impact of the threshold in the utility functions

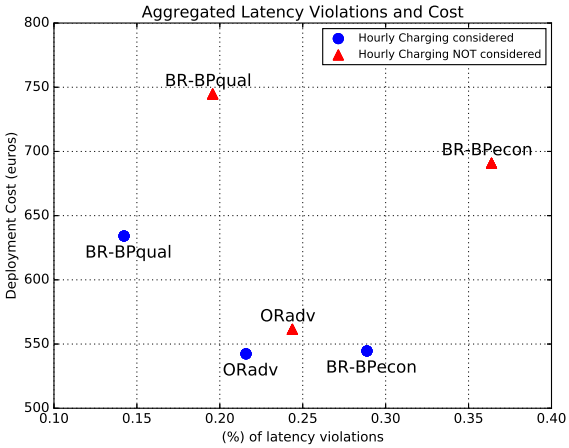


Fig. 11 Hourly charging vs. charging per time unit

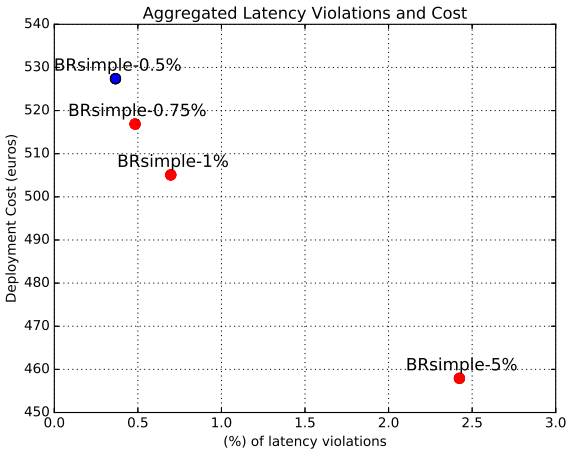


Fig. 12 Impact of reward bound

utility function, where $BR - BP_{qual}$ achieves the least percentage of latency violations, however the OR_{adv} and BR_{simple} yielded slightly higher (i.e., up to 10 euros) deployment cost.

6. $BR-BP_{econ}$ policy is the result of the application of extra probabilistic verification steps and the specification of a bound in the probability of latency violation on top of BR_{econ}. Hence, $BR-BP_{econ}$ can be deemed as a conservative version of the BR_{econ} policy regarding the reduction of the number of VMs. If we observe the behavior of these two policies, in almost all the cases, the application of the extra quantitative analysis results in a increase in the deployment cost and a reduction in the latency violations, as expected. An exception is when the *NWE* utility function is used (see Figure 9(c)), where being conservative does not pay off. This is attributed to the fact that, if at a specific step a policy does not add the required VMs, may be forced to proceed to a series of additions later with potentially higher cost and after having experienced latency violations.
7. Using the *WE4060* mitigates the need for the additional probabilistic verification. This is attributed to the fact that the specific utility function configuration inherently leads to decisions that aim to avoid latency violations to a larger extent compared to the other WE utility function configuration. This also explains the behavior of BR_{simple} in Figure 9(d).

6.3 Sensitivity Analysis

We have proceeded to additional experiments to analyze the sensitivity of our approach to their parameters and fine-tune the elasticity decision maker. The first experiment investigates the impact of not using the threshold condition in the utility functions in Eq. (3)-(5). Figure 10 shows the results for Eq. (5), using the OR_{adv} policy; the blue circles correspond to the original Eq. (5), while the red triangles to its modifications. The observation is that not using the threshold leads to worse trade-offs and severe degradation of the performance, as shown from the high number of latency violations.

In the next experiment, we deactivate the charging per hour and we assume a pricing scheme where charging is per time unit. This modification leads to higher deployment costs with almost the same or worse percentages of response latency violations than when the hourly charging is enforced. Figure 11 presents three representative cases, using the utility function of Eq. (3), where it is shown that the resulted trade-offs are inferior. This is attributed to the fact that, in an hourly-based charging model, VMs are kept for longer periods without extra cost.

In all experiments thus far, we have set the sub-optimal reward bound to +0.5% of the optimal reward. Various other values have been also tested. Figure 12

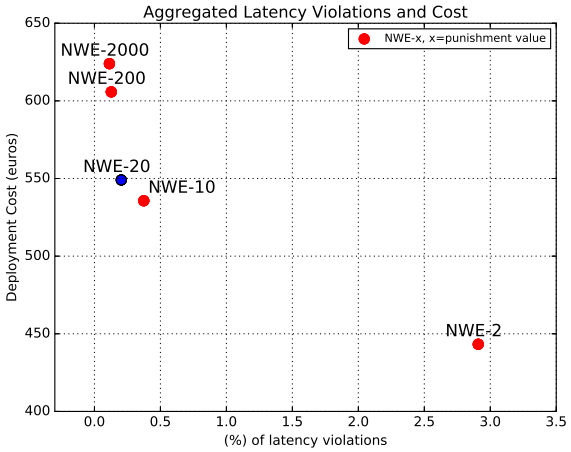


Fig. 13 Impact of the punishment value in *NWE*

presents the results for the BR_{simple} policy and the utility function of Eq. (3). In general, different trade-offs can be achieved by simply tuning this bound. Compared to the 0.75% case, our default choice achieves 23% less latency violations with just 2.6% increase in the deployment cost.

Finally, Figure 13 investigates the impact of the punishment value in Eq. (4), using the OR_{adv} policy. Setting the same punishment value as in Eq. (3) (i.e. 2), keeps the cost at low levels (443.17 euros), however the latency violation percentage is severely increased (2.91%). Setting the punishment value too high (i.e. 2000), has a negative impact on the cost (623.95 euros). 20 was selected as the most appropriate one as it achieves a more balanced trade-off (0.2% latency violations, 549.06 euros deployment cost).

Further sensitivity analysis for OR_{adv}, which includes the decision frequency, the prediction accuracy and the latency threshold, is presented in our previous work [37]; these results are transferred to the techniques presented hereby as well.

7 Conclusions and Future Work

This work presents a principled approach to horizontal scaling for elastic NoSQL databases deployed on cloud infrastructures. Two contradicting objectives are pursued, namely cost minimization and avoidance of latency violations. The underlying model is a MDP, on top of which probabilistic model checking is applied to provide guarantees regarding latency violations. The main novelty of this work is the direct consideration of the monetary cost of the cloud deployment in the decision making process, coupled with the proposal of specific decision making policies, which outperform our previously proposed policies in [37, 38] and shown to be capable of offering configurable trade-offs between cost and latency. Our proposal is generic and is applicable to any elastic application exhibiting unpredictable performance where no analytical models can be derived, and there exist significant transient periods after each elasticity action.

The main directions for future work are twofold. First, to elaborate on the support of heterogeneous clusters, which is briefly mentioned in Section 4.2. Second, to support additional forms of elasticity and adaptations, such as vertical scaling, migration and reconfiguration. For both directions, the straightforward extension is to develop models with distinct states for each combination of type of clusters, locations and configurations. However, this would lead to an explosion of the size of the model and model checking would become inefficient. Currently, our models are processed in a few seconds on a modern machine, but much complicated models would require many minutes, which is longer than a reasonable decision step. Thus, the main challenge regarding the two directions above is to tackle the scalability problems that are involved.

References

1. Ahmad Al-Shishtawy and Vladimir Vlassov. Elastman: elasticity manager for elastic key-value stores in the cloud. In *ACM Cloud and Autonomic Computing Conference, CAC '13, Miami, FL, USA - August 05 - 09, 2013*, page 7, 2013.
2. Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212. IEEE, 2012.
3. FJ Almeida Morais, F Vilar Brasileiro, R Vigolvino Lopes, R Araujo Santos, Wade Satterfield, and Leandro Rosa. Autoflex: Service agnostic auto-scaling framework for iaas deployment models. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 42–49, 2013.
4. Adnan Ashraf, Benjamin Byholm, and Ivan Porres. Cramp: Cost-efficient resource allocation for multiple web applications with proactive scaling. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 581–586, 2012.
5. Lakshmi N. Bairavasundaram, Gokul Soundararajan, Vipul Mathur, Kaladhar Voruganti, and Kiran Srinivasan. Responding rapidly to service level violations using virtual appliances. *SIGOPS Oper. Syst. Rev.*, 46(3):32–40, 2012.

6. Sean Kenneth Barker, Yun Chi, Hakan Hacigümüs, Prashant J. Shenoy, and Emmanuel Cecchet. Shuttledb: Database-aware elasticity in the cloud. In *11th International Conference on Autonomic Computing, ICAC '14, Philadelphia, PA, USA, June 18-20, 2014.*, pages 33–43, 2014.
7. Leander Beernaert, Miguel Matos, Ricardo Vilaça, and Rui Oliveira. Automatic elasticity in openstack. In *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, page 2. ACM, 2012.
8. Javier Cámara, Gabriel A. Moreno, and David Garlan. Stochastic game analysis and latency awareness for proactive self-adaptation. In *SEAMS*, pages 155–164, 2014.
9. Valeria Cardellini, Emiliano Casalicchio, Francesco Lo Presti, and Luca Silvestri. Sla-aware resource management for application service providers in the cloud. In *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, pages 20–27. IEEE, 2011.
10. Maria Chalkiadaki and Kostas Magoutis. Managing service performance in the cassandra distributed storage system. In *IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, December 2-5, 2013, Volume 1*, pages 64–71, 2013.
11. G. Copil, D. Moldovan, Hong-Linh Truong, and S. Dustdar. On controlling cloud services elasticity in heterogeneous clouds. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 573–578, 2014.
12. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. Sybl: an extensible language for controlling elasticity in cloud applications. In *IEEE International Symposium on Cluster Computing and the Grid (to appear), CCGRID*, volume 13, 2013.
13. Emanuel Ferreira Coutinho, Flávio Rubens de Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonçalves Gomes, and José Neuman de Souza. Elasticity in cloud computing: a survey. *annals of telecommunications-annales des télécommunications*, pages 1–21, 2015.
14. Francisco Cruz, Francisco Maia, Miguel Matos, Rui Oliveira, João Paulo, José Pereira, and Ricardo Vilaça. Met: workload aware elasticity for nosql. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 183–196. ACM, 2013.
15. Ariel da Silva Dias, Luis Hideo Vasconcelos Nakamura, Júlio Cezar Estrella, Regina Helena Carlucci Santana, and Marcos José Santana. Providing iaas resources automatically through prediction and monitoring approaches. In *IEEE Symposium on Computers and Communications, ISCC 2014, Funchal, Madeira, Portugal, June 23-26, 2014*, pages 1–7, 2014.
16. Pierangelo Di Sanzo, Diego Rughetti, Bruno Ciciani, and Francesco Quaglia. Auto-tuning of cloud-based in-memory transactional data grids via machine learning. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 9–16. IEEE, 2012.
17. Xavier Dutreilh, Nicolas Rivierre, Aurélien Moreau, Jacques Malenfant, and Isis Truck. From data center resource allocation to control theory and back. In *IEEE CLOUD*, pages 410–417, 2010.
18. Sourav Dutta, Sankalp Gera, Akshat Verma, and Balaji Viswanathan. Smartscale: Automatic application scaling in enterprise clouds. In *IEEE CLOUD*, pages 221–228, 2012.
19. Hector Fernandez, Guillaume Pierre, and Thilo Kielmann. Autoscaling web applications in heterogeneous cloud infrastructures. In *2014 IEEE International Conference on Cloud Engineering*, pages 195–204, 2014.
20. V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113, 2011.
21. Soguy Mak Karé Gueye, Noel De Palma, Éric Rutten, Alain Tchana, and Nicolas Berthier. Coordinating self-sizing and self-repair managers for multi-tier systems. *Future Generation Comp. Syst.*, 35:14–26, 2014.
22. Rui Han, Moustafa Ghanem, Li Guo, Yike Guo, and Michelle Osmond. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. *Future Generation Comp. Syst.*, 32:82–98, 2014.
23. Rui Han, Li Guo, Moustafa M Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 644–651. IEEE, 2012.
24. Zhenhua Han, Haisheng Tan, Guihai Chen, Rui Wang, Yifan Chen, and Francis Lau. Dynamic virtual machine management via approximate markov decision process. *arXiv preprint arXiv:1602.00097*, 2016.
25. Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Comput-*

ing (ICAC 13), pages 23–27, Berkeley, CA, 2013. USENIX.
26. Daniel Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.
27. Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. Optimal cloud resource auto-scaling for web applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 58–65. IEEE, 2013.
28. Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using kalman filters. In *Proceedings of the 6th International Conference on Autonomic Computing, ICAC 2009, June 15-19, 2009, Barcelona, Spain*, pages 117–126, 2009.
29. Vangelis Koukis, Constantinos Venetsanopoulos, and Nectarios Koziris. ∼okeanos: Building a cloud, cluster by cluster. *IEEE Internet Computing*, 17(3):67–71, 2013.
30. Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: probabilistic model checking for performance and reliability analysis. *SIGMETRICS Perform. Eval. Rev.*, 36(4):40–45, 2009.
31. Harold C Lim, Shivnath Babu, and Jeffrey S Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, pages 1–10. ACM, 2010.
32. Ming Mao and Marty Humphrey. Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*, pages 67–78, 2013.
33. Paul Marshall, Kate Keahey, and Timothy Freeman. Elastic site: Using clouds to elastically extend site resources. In *CCGRID*, pages 43–52, 2010.
34. Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In *Proceedings of the Joint Meeting of the European Software Engineering Conference and the Symposium on Foundations of Software Engineering (ESEC/FSE)*, 2015.
35. Gabriel A Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. Efficient decision-making under uncertainty for proactive self-adaptation. 2016.
36. Athanasios Naskos, Anastasios Gounaris, and Spyros Sioutas. Cloud elasticity: A survey. In *ALGOCLOUD*, 2015.
37. Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. Dependable horizontal scaling based on probabilistic model checking. In *CCGrid*. IEEE, 2015.
38. Athanasios Naskos, Emmanouela Stachtiari, Panagiotis Katsaros, and Anastasios Gounaris. Probabilistic model checking at runtime for the provisioning of cloud resources. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, pages 275–280, 2015.
39. Fawaz Paraiso, Philippe Merle, and Lionel Seinturier. Managing elasticity across multiple cloud providers. In *Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds*, pages 53–60, 2013.
40. Diego Perez-Palacin, Raffaela Mirandola, and Radu Calinescu. Synthesis of adaptation plans for cloud infrastructure with hybrid cost models. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, Verona, Italy, August 27-29, 2014*, pages 443–450, 2014.
41. Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.
42. Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. Slo-aware deployment of web applications requiring strong consistency using multiple clouds. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 860–868. IEEE, 2015.
43. Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 500–507. IEEE, 2011.
44. Damian Serrano, Sara Bouchenak, Yousri Kouki, Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, Pierre Sens, et al. Towards qos-oriented sla guarantees for online cloud services. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2013*, pages 0–0, 2013.
45. Dimitrios Tsoumakos, Ioannis Konstantinou, Christina Boumpouka, Spyros Sioutas, and Nectarios Koziris. Automated, elastic resource provisioning for nosql clusters using tiramola. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 34–41. IEEE, 2013.

46. Luis M Vaquero, Daniel Morán, Fermín Galán, and Jose M Alcaraz-Calero. Towards runtime reconfiguration of application control policies in the cloud. *Journal of Network and Systems Management*, 20(4):489–512, 2012.
47. Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers*, 16(1):7–18, 2014.