Tsagkaropoulos et al. Journal of Cloud Computing: Advances, Systems and Applications (2021) 10:45
https://doi.org/10.1186/s13677-021-00255-5

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Severity: a QoS-aware approach to cloud application elasticity

Andreas Tsagkaropoulos[1,2*], Yiannis Verginadis[1,3], Nikos Papageorgiou[1,2], Fotis Paraskevopoulos[1,2], Dimitris Apostolou[1,4] and Gregoris Mentzas[1,2]

Abstract

While a multitude of cloud vendors exist today offering flexible application hosting services, the application adaptation capabilities provided in terms of autoscaling are rather limited. In most cases, a static adaptation action is used having a fixed scaling response. In the cases that a dynamic adaptation action is provided, this is based on a single scaling variable. We propose Severity, a novel algorithmic approach aiding the adaptation of cloud applications. Based on the input of the DevOps, our approach detects situations, calculates their Severity and proposes adaptations which can lead to better application performance. Severity can be calculated for any number of application QoS attributes and any type of such attributes, whether bounded or unbounded. Evaluation with four distinct workload types and a variety of monitoring attributes shows that QoS for particular application categories is improved. The feasibility of our approach is demonstrated with a prototype implementation of an application adaptation manager, for which the source code is provided.

Keywords: Cloud computing, Cloud applications, Cloud application adaptation

Introduction

Cloud services are increasingly used with the total amount spent on public cloud services, globally, predicted to surpass $300 billion in 2021 [1]. A diverse range of applications can be deployed on machines installed in public, private and hybrid clouds, leveraging available APIs of cloud providers and standards such as TOSCA (Topology and orchestration specification for cloud applications) [2]. For some cloud applications, the initial deployment might be enough to ensure that the application will operate correctly indefinitely, as the workload is expected to be static and predictable. In most cases however, the load is fluctuating and thus requires support for elasticity from the part of the application. Elastic applications should be able to respond to the continuous need for adaptations, adding, removing or modifying a number of VM (Virtual Machine) instances to appropriately handle incoming load. Towards this direction, major cloud providers offer support for automated application scaling which is typically controlled by a set of predefined rules.

Cloud application elasticity is an active research domain, especially if we consider the need to handle the extended cloud computing, which includes fog and edge devices. Algorithms exploiting a range of techniques, from machine learning to control theory methods, have been developed, aiming to provide timely application adaptations at low cost. From the modelling perspective, software architecture paradigms which emphasize on scalability and adaptivity, culminating to the Functions as a Service (FaaS) paradigm have gained considerable attention over the last years. These recent developments, coupled by the surge in popularity of cloud applications, have created a pressing need to properly cope with workload fluctuations and adequately handle heterogeneous cloud resource allocation that might even involve edge devices.

* Correspondence: atsagkaropoulos@mail.ntua.gr
[1]Information Management Unit (IMU), Institute of Communication and Computer Systems, Athens, Greece
[2]National Technical University of Athens (NTUA), Athens, Greece
Full list of author information is available at the end of the article

The mechanisms supporting cloud elasticity which are available today, can be broadly divided in three categories: i) manual decision making; ii) automatic adaptations; and iii) hybrid solutions. To the best of our knowledge, most of the developed mechanisms either require some input from the DevOps in the form of event-condition-action rules – and do little more than applying them – or rely on complex solutions inspired from the fields of control-theory, queueing-theory and machine-learning [3]. The knowledge acquired from machine-learning – based algorithms is not easily transferable to other domains (the transfer learning problem [4]) and they inherently lack transparency in the way the decision-making takes place, a fact that may hinder their adoption in production systems at this stage. Mechanisms requiring manual input are the most popular solution among the available providers.

In this work we present 'Severity', a novel algorithmic approach aiding the adaptation of cloud applications. Based on the input of the DevOps, our approach detects situations, calculates their Severity and proposes adaptations which can lead to better application performance. Our approach is complemented by a prototype software system, which uses Severity to characterize the current load and produce the necessary adaptation actions. Unlike other similar approaches, the triggering of adaptation actions, as well as the adaptation actions themselves are provided in an easily understandable form and it is possible to log precisely the cause and the effect of each adaptation recommendation of the application. Our approach mainly focuses on the usage of horizontal scaling.

In this paper, we aim to answer the following research questions: i) How does our methodology compare to other methodologies enabling application elasticity? ii) What is the deviation of our approach from the optimal solution? iii) How does Severity compare to commercial approaches, in terms of satisfaction of QoS attributes?

The remainder of this work is organized around the definition of 'Severity' and the techniques which are based on it in 5 sections. Section 2 includes a view of the current state of the art concerning the adaptation of cloud applications. Section 3 includes the definition of elasticity rules and introduces the concept of Severity. Section 4 contains the definitions of adaptation techniques which can be used to control the (horizontal) scaling of an application. Section 5 demonstrates the calculation of Severity and the resulting adaptation in an illustrative scenario. Section 6 presents the feasibility of using Severity in a real-world platform monitoring system. In Section 7, a comparative evaluation of the adaptation techniques described in Section 4 is presented. Section 8 outlines the findings of the paper, while also proposing future research directions. Finally, Section 9 concludes this work.

Cloud application elasticity

Decision-making approaches for cloud autoscaling systems are based in general either on rule-based control, or Control Theory and Search-based optimization [5]. In the following subsections we position our approach with respect to other works from these fields. Our emphasis is on applications with workloads consisting of parallel tasks which can be finely partitioned between processing instances, e.g., using a load balancer. This marks a difference from other works which consider an internal application structure (e.g complex workflows as in [6]). We assume that our simulated application uses a single virtualization layer, and that no synchronization issues as those investigated in [7] appear. Moreover, unlike some works which consider resource contention between cloud components (e.g [8]), we consider that ample cloud resources are available, and that no resource saturation occurs in the cloud.

Rule-based and control-theoretic adaptation approaches

The rule-based adaptation approach is one of the simplest and more intuitive approaches which can be followed to scale a cloud application. Rule-based adaptations rely on the expertise of a DevOps to define the variables which should be monitored, and the thresholds which should be respected (a priori knowledge). The rules should be carefully tuned in order to include all variables which can influence the deployment. While some adaptation systems only use some adaptation attributes for input (as indicated in [5]) such as CPU or the response time, the system which is proposed in this work can work with any number and type of measurable attributes.

Control theoretic approaches (e.g [9–11]) are based on traditional control theory but are occasionally enhanced with extensions. They are characterized by their dynamicity and low latency. However, the configuration of the control loop should be performed by an expert in order to prevent waiting for the system to stabilize after multiple iterations. Rule-based approaches hold an advantage over pure control-theoretic approaches on simplicity and clarity, and thus domain experts can more easily transfer their knowledge to the systems.

Gandhi et al. [3] describe a technique based on Kalman filtering, which estimates the parameters of a queuing model representing the application. The estimated values are used to create scaling directives, providing auto-scaling capabilities to the platform. When an abrupt change in its time-series representation is detected, a scaling event is transmitted. The authors evaluated the performance of their algorithm and found it superior to threshold-based rules working with static percentages, adding or removing one VM instance when the threshold (upper or lower respectively) has been

violated. Using our approach, scaling actions can be more varying and detailed in their response, than simply adding or removing a single VM instance. Besides response time, we can use any number of attributes to feed our techniques. Lorido-Botran et al. [12] comment on queueing theory models used to horizontally scale an application, that they suffer from being tightly bound to the workload and the processing infrastructure for which they have been created. As a result, they need to be recalculated when either change.

Arkian et al. [13] propose Gesscale, a control-theory inspired autoscaling approach, based on the measurement of the maximum sustainable throughput. The estimation of the results of a scaling action in Gesscale is based on the existence of a performance model. When the performance is better than expected, multiple processing instances can be removed, while when it is worse, a single processing instance is added. They use a single composite metric (maximum sustainable throughput - MST) to guide their autoscaling model. MST is calculated based on the maximum network delay between nodes, the throughput of a single node, and the parallelization inefficiency. Using their methodology, they demonstrate superior performance compared to algorithms which are latency-unaware, and/or use only the cpu consumption as an indication of the intensity of the workload. In practice this approach still uses one strictly defined (albeit composite) metric to guide scaling. Instead, our approach combines any number of arbitrary metrics to obtain better results rather than using individual metric values to scale. Moreover, the scaling algorithms we define allow more than one instances to be added as necessary which reduces the number of reconfigurations.

In [14] the approaches of Amazon and Google concerning scaling are described, Target Tracking and Step Scaling, and Multiple Zones and Horizontal-Pod Autoscaling (Kubernetes) respectively. These tools can be divided into two algorithmic categories, the first containing Multiple Zones and Step scaling, and the second containing the Kubernetes Horizontal-Pod Autoscaling and Target Tracking. In the first category of tools (which is also encountered in other major providers, such as Microsoft Azure [15] and Oracle Cloud Infrastructure [16]), the DevOps should either enter a number of rules that scale out/in the application by a predefined number of instances (or a percentage of the active instances). Unfortunately, while this approach is simple, it requires considerable input from the DevOps. Tools belonging in the second category are more sophisticated, requiring the creation of a control-loop that will perform scaling automatically to attain a specific threshold value. While Amazon Target Tracking supports only one metric in the Control Loop, the Kubernetes horizontal pod auto-scaler (HPA) can support multiple metrics.

In the same work [14], a custom approach to scaling using the 'dynamic-multi level' (DM) method is outlined, combining predictive elements with a control loop to direct the scaling of the platform. Using a variety of workloads and benchmarking metrics, an evaluation against a real system was carried out, and their approach was found to be better than approaches which are used by leading cloud vendors in many scenarios. However, only one threshold value was used for all algorithms, and a default VM instantiation delay of 30 s was assumed. The adaptation techniques proposed in our work were evaluated using four VM spawn delay intervals, as well as six combinations of thresholds to detect variation in their performance. Additionally, the workload patterns which are used in our evaluation are more radically changing compared to those provided in [14] (in terms of the rate of change of the absolute values of the workload), stress-testing the performance of all techniques.

In [17], an extension to the Kubernetes HPA algorithm is discussed, evaluating the use of a constant absorbing small fluctuations of the workload. We claim that in cases where scaling is performed using multiple metrics, one or more of the performance criteria of the application can be improved, when all of the available values of the monitoring metrics are used (rather than only the maximum value or only one metric value as is the case in Kubernetes HPA and Amazon TTS, respectively). Additionally, we provide an approach which requires less input from the DevOps and allows hybrid algorithms – for example a control loop activated by rule thresholds as illustrated in section 4.5.

Another interesting approach is followed by Lorido-Botran et al. [18], who thoroughly describe the idea of modifying the thresholds which are employed in rule-based systems to obtain a better response. They support that when no service-level objective violations are detected within a time frame, the scale-in and scale-out rule thresholds should converge to higher and lower values respectively to improve the responsiveness of the system when high workload is encountered. The evaluation of their algorithm is performed using a single, highly variable workload trace, and two benchmarking criteria (service-level objective violation and cost). Approaches similar to those introduced in [14, 17, 18] are complementary to our approach and can be used to provide an enhanced yet more complex system.

Vaquero et al. [19], Galante et al. [20] and Copil et al. [21, 22] have proposed rule-based frameworks, which either rely on user input to calibrate the adaptation actions by manually setting the scaling action as in [19, 20] or always using the same adaptation event (e.g add one VM instance) to keep the desired monitoring attributes

to acceptable levels as in [22]. Using these frameworks, the user should manually detail all the situations for which adaptation will be required. However, this process is error-prone and nevertheless requires the constant attention of the DevOps. In the case of Ferretti et al. [23], the ability to implement scaling decisions adding or removing more than one instances is supported, however no information is provided on whether the number of instances (de) allocated can change at runtime without the intervention of the DevOps. Our approach aims to waive the requirement from a user to frequently change the response of the system, as the user needs only to specify basic thresholds with a generic action once. Then, we can measure the violation of these thresholds and automatically derive an adaptation action.

In [24], Trihinas et al. enhance the basic adaptation support offered by the previous rule-based systems, by offering AdaFrame, a library to support resource-based elasticity controllers. AdaFrame improves the results of rule-based systems by adapting a cooling-down period between successive adaptations, through the analysis of the statistical properties of a monitoring metric stream, e.g., CPU utilization. Thus, scaling out and scaling in actions are less likely to occur on sudden bursts, and occur faster in the case of increased 'regular' workload. This approach is complementary to ours, as it improves the triggering of the autoscaling loop.

Dutreilh et al. [25] have explored both threshold-based rules and Q-learning, concluding that Q-learning is superior, given enough training. Two of the techniques which are examined in Sections 4.6 and 4.7 are simplifications of Q-learning, with the absence of feedback. Unlike Q-learning though, our approach, benefits from being usable without extensive training or requiring the definition of a complex reward function. Besides, we describe a mechanism to ascertain the Severity of a situation, similar to the reward function employed in Q-learning, which can be used as input for a multitude of algorithms, one of which can also be Q-learning.

Ali-Eldin et al. discuss in works [26, 27] elasticity controllers based on a generic model of queueing theory, the G/G/N queue. In work [27] they consider workloads which can be queued and then be appropriately serviced by tuning the number of VMs according to the requests which should be serviced. Their approach allows a service to remain operational even under heavy load, by limiting the queued requests. The availability of a buffer to queue requests is not present in their previous work [26]. The principle behind the scaling of the application in their approach is similar to the algorithms which we propose. Moreover, our approach can be used in conjunction with their proposal, since through Severity we shall provide a means to scalarize a set of metric values to be able to use a single-metric based controller such as the one which is proposed in their work. We concentrate on the ability to extract more information from rules involving multiple metrics, however this does not preclude the use of the advanced techniques presented in these works (e.g different combinations of reactive and/or proactive scaling-up and scaling-down).

In [28] the authors propose Chameleon, a hybrid, proactive autoscaling mechanism, evaluating its performance using realistic workloads and works suggested in the state of the art. CPU utilization and request rate are mainly used to estimate the workload of an application and guide autoscaling. Chameleon combines forecasting methods and realtime monitoring to enable proactive and reactive scaling decisions. It uses thresholds for both reactive and proactive scaling decisions, however the service demand estimation component also uses Kalman filter, regression and optimization estimators (among others) to estimate the time required for a request. Chameleon is extensively evaluated against other approaches suggested in the autoscaling literature and found to outperform them by a large margin. Our approach enables the use of more metrics if necessary, including custom metrics. As such, we argue that it can enrich approaches such as Chameleon to consider additional context factors (metrics) for their autoscaling algorithms.

Search-based optimization adaptation approaches

Search-based optimization approaches comprise another main category of decision-making approaches used by Self-aware and Self-adaptive Cloud Application Systems (SSCAS) [5]. Using the classification of Chen et al. [5], search-based techniques include dynamic programming, genetic algorithms, reinforcement learning and integer linear programming among others. By definition, all of these techniques are based on traversing the search space of solutions using a specific algorithm, attempting to optimize one or more criteria. However, the exponential number of solutions which should be explored when considering a number of attributes and actions which can be optimized results in training or execution times which are unacceptable. Also, while these techniques require less work from the side of the domain expert – as a lot of information is learnt at runtime – they need more time to converge. Additionally, when finished, the actions learnt are highly specific to the problem solved – meaning that unless they can be translated to a set of rules/statements, no knowledge can be transferred in a case of a new but different instance of the problem.

The work of Ramirez et al. [29] describes an autoscaling mechanism which considers two virtualization layers (VMs and containers) to deliver the required quality of service. Quality of service is calculated based on the number of requests which can be serviced. Five different

techniques to determine the number of VMs and containers for a workload are evaluated, three of which traverse the configuration space (number and type of containers in VMs) to find an appropriate solution (the others use heuristics). They demonstrate that appropriately handling scaling using two virtualization layers results in reduced cost. The techniques we describe can be used in parallel with such approaches although we focus on applications which exploit a single virtualization layer (or use the assumption of one container instance in one virtual/physical machine). Moreover, we allow multiple metrics to influence the decision of scaling.

In [30] the authors compare two functionality modes of the Kubernetes Cluster Autoscaler. The Kubernetes Cluster Autoscaler is a component responsible to allocate new processing nodes to host Kubernetes pods when this is necessary. The first functionality involves using nodes from a single node pool (identical nodes - CA) while in the second multiple node pools are used (allowing differently-sized nodes to be spawned – CA-NAP). They conclude using standard autoscaling metrics that CA-NAP is overall superior to the CA, although no significant cost benefits are observed. In the evaluation of this work, we do not consider using nodes with different processing capacity. However, our techniques can be generalized to use processing nodes offering a fraction of the performance of a normative processing node.

GKE Autopilot [31] offers an advanced autoscaling approach, capable of vertical and horizontal autoscaling. The main emphasis of this work is on vertical scaling, setting the appropriate resource limits for each processing node. Autopilot can set these limits, even if no user input is provided. Autopilot manages to greatly reduce slack (unused resources) using either statistical or machine learning techniques. However, the configuration of statistical recommendations is tuned for long running services, which might not be optimal. Besides, its machine learning recommender has the advantage that it can output easily explainable recommendations on the resource limits of a processing node. The part related to horizontal scaling resembles the algorithm which is used by the Kubernetes Horizontal Pod Autoscaler.

AWS [32] offers another autoscaling approach which is based on predicted data about the application. It tries to attain a target utilization level, based on monitoring metrics (it does not however currently support custom metrics). It uses machine learning models trained in Amazon, based on billions of data points. To the best of our knowledge these machine learning models are not publicly available. Notwithstanding, it is difficult to train ML models of a comparable size without access to the data, algorithms and processing infrastructure used by Amazon. Our approach does not need any training, the adaptations created by it can be readily traced to the original monitoring observations and can be used in addition to the presence of a forecasting mechanism. Our techniques extract added value from domain expert knowledge while retaining the simplicity of threshold rules. We introduce a thoroughly documented, open and modular approach which - thanks to the scalarization realized through Severity - can use ideas present in any existing horizontal scaling adaptation technique using metrics (as Severity itself can be considered a metric).

Cloud application elasticity with elasticity rules

Elasticity rules

We define elasticity rules as directives which indicate firstly the QoS limits of normal operation of an application, and secondly the horizontal elasticity action which should be taken to accommodate the needs of the application when these limits are trespassed, scaling in or scaling out. The QoS limits can be specified in terms of any measurable metric, including custom metrics. Rules are assumed to be entered by a DevOps who possesses significant experience and knowledge on the application which is deployed and monitored. Contrary to static rules which specify one concrete set of conditions, and one concrete set of actions, the proposed elasticity rules require less input for their definition. The DevOps should specify the QoS conditions which trigger the rule, but in the action part of the rule, only the scaling direction is required and not the number of instances which should be added/removed. This allows a flexible response action, which may be decided using a variety of techniques as demonstrated in section 4. With elasticity rules, the triggering conditions of a rule and the adaptation - the concrete actions - are separated conceptually; our approach assumes that a DevOps is primarily interested in defining the criteria indicating that the application functions correctly, rather than the exact adaptation action which will be followed.

The format of an elasticity rule can be found in Listing 1.

Listing 1 Elasticity rule format:

$$\textbf{\textit{For component\_id}} = component\_id$$

$$\textbf{\textit{if}}\,(attribute\_1) < \; attribute\_1 \;\; value \;\; \textbf{\textit{and}} \;\; (attribute\_2) < attribute\_2 \;\; value \;\; \textbf{\textit{and}} \;\; \ldots \;\; \textbf{\textit{and}} (attribute\_n) < \; attribute\_n \;\; value$$

$$\textbf{\textit{within Timewindow}} = Timewindow \;\; \textbf{\textit{and}} \;\; Cooldownperiod \;\; \textbf{\textit{has passed from previous adaptationthen}} \;\; Scale\_out/Scale\_in$$

Any number of QoS attributes connected with the "AND" logical operator can be entered. While we do not allow "OR" logical operators to be used alongside "AND" logical operators, multiple elasticity rules can be enforced in parallel. Additionally, non-bounded attributes, e.g., response time, are supported, provided that a threshold is set by the DevOps. Furthermore, the

DevOps defines the time-window over which this rule is calculated (e.g., 10 min), and the cooldown period which should elapse between two triggerings of the rule.

An instantiated example of an elasticity rule appears in Listing 2.

Listing 2 Elasticity Rule example

For component_id = *VideoTranscoder*
if $AverageCPU_{cluster}$>70%***and*** $AverageRAM_{cluster}$>70%
within Timewindow=10 *minutes*
and 30 *minutes* ***have passed from previous adaptations then*** *Scale_out*

When the thresholds set by the DevOps for the monitoring attributes expressed in an elasticity rule are violated, a violating *situation* is detected, and an elasticity rule is triggered. Onwards, we will refer to a violating situation simply as a 'situation'.

Situation severity

Once the thresholds of an elasticity rule are violated, an associated situation is detected. To assist the decision on the most appropriate number of instances which should be added to or be removed from the application, we assess the 'Severity' of the situation, which quantifies the rough magnitude of the violation of the thresholds of the attributes used in the elasticity rule. The values of all violating attributes are used, and weights are assigned to each of them to indicate their relative importance. Higher values of Severity indicate that more pronounced changes to the application should be made (i.e., more VMs should be added/removed).

The Severity of any detected situation $\mathbf{V}_{\text{violating}} = (v_1, v_2, \ldots, v_n)$ is determined as shown in Eq. 1.

Equation 1 Calculation of the Severity of a situation

$$Severity(\boldsymbol{V}_{violating}) = \sqrt{\sum_{i=1}^{n} w_i (Normalized(v_i))^2} \quad (1)$$

where $\mathbf{v_i}$ are the individual, threshold violating QoS attribute values comprising the particular situation, $\mathbf{w_i}$ are their respective weights and $\mathbf{n}$ is the number of attributes included in the triggered elasticity rule. For each of the $\mathbf{v_i}$ svalues it is assumed that $v_i \in [0, 1]$.

While the definition of Severity allows for the usage of different weights for each of the attributes being evaluated, in the remainder of our work we assume for simplicity that all weights are equal to 1. Following the definition of Severity (Eq. 1), the maximum Severity value for a situation is observed when all attributes have reached their maximum normalized values, i.e., 1 and is equal to $\sqrt{n}$.

Having chosen the weight values for each attribute, the calculation of Severity relies on obtaining the normalized values for each of the attributes. For each attribute $\mathbf{v_i}$ - threshold $\mathbf{t_i}$ pair in the rule, the normalization formula in Eq. 2 is used in cases of attributes that need to be greater than their threshold and Eq. 3 is used in case of attributes that need to be less than their threshold.

Equation 2 Variable normalization in the greater-than case

$$Normalized(v_i) = \frac{abs(v_i - t_i)}{abs(\text{maximum}(\text{attribute}_i) - t_i)} \quad (2)$$

Equation 3 Variable normalization in the less-than case

$$Normalized(v_i) = \frac{abs(v_i - t_i)}{abs(t_i - \text{minimum}(\text{attribute}_i))} \quad (3)$$

Equations 2 and 3 are applicable in the case of attributes which are bounded. In the case of unbounded attributes – for example response time, the denominator of Eq. 2 and Eq. 3 is unknown, and therefore the normalized value is not computable. In such cases, we can estimate the unknown or unavailable bounds of the attribute using past observations. We assume that each attribute follows an arbitrary distribution and that the attribute – random variable is integrable, has a finite expected value μ, a finite non-zero variance σ^2 and a standard deviation σ. For the estimation of the bounds we use Chebyshev's equation [33].

We consider that determining that 96% of the samples of the attribute are within an upper and a lower bound, provides us with an adequate estimation of the maximum and the minimum value respectively. In this case, only 4% of the samples will be outside these boundaries. Substituting this probability value in the left handside of the equation, and solving for t we determine that $t = 5\sigma$. As a result, we can conclude that the contrapositive argument, i.e., that all samples of a distribution will be contained inside the boundaries with a probability of 96%, is true as long as the samples are within 5 standard deviations of its mean value.

To illustrate, using the example of response time – which does not have an upper bound – let us assume that current observations for this attribute indicate an expected value $\mu = 200$ msec, with a standard deviation of 30 msec. Then, the probability of measuring a response time X, being retarded more than $5 \cdot 30msec = 150\ msec$ from the expected value (200 msec) is less than 4%. We can then estimate the upper bound of the distribution to be 200 + 150 = 350 msec with 96% probability.

The expected value and the standard deviation of the distribution is calculated as the arithmetic mean over a window of the last z samples of the distribution – a number which can be configurable. The greater the value of z, the more the arithmetic mean will approach the expected value of the distribution (provided that the

distribution is unchanged). The smaller the value of z, the more flexible the bounds are to changes in the distribution of unbounded variables. 'Upper' and 'lower' bounds are updated dynamically using a sliding event window.

Severity zone calculation

Although Severity provides an assessment of the intensity of a situation, in some cases it is meaningful to group situations by their Severity. Considering that a detected situation with p attributes is represented as a point in p-dimensional space by $\mathbf{V}_{\text{violating}}$, situations having similar Severity values form circular annuli, spherical shells or hyper-spherical shells (depending on whether $p = 2$, $p = 3$, or $p \geq 4$ respectively). These regions are called 'Severity zones' and are used by the Simple severity zones and Relative severity zones techniques presented below.

The rationale behind both of these techniques is that similar Situations in terms of Severity should result in same adaptation actions. To obtain 'Severity zones', we divide the real number interval $[0, \sqrt{n}]$ reflecting all possible Severity values for a given set of metrics into m equal sub-intervals. Each such sub-interval is a Severity zone (Table 1).

Zones containing situations with Severity values with numbers closer to 0 will result in milder adaptation actions, while Severity zones closer to $\sqrt{n}$ (the maximum value of Severity) will result in more instances being added to/removed from the application. Choosing higher values for m indicates that finer-grained adaptation actions are required. On the other hand, choosing lower values for m increases the amount of historical data available for each adaptation action.

We make adaptation decisions under the assumption that the Severity value calculated from a random situation can belong to each zone with equal probability – in order not to bias the triggering of a particular adaptation decision – and thus obtain results which are relevant to the situations included in the particular Severity zone. To satisfy this requirement, we need to define all Severity zones to have equal area (or volume, or hypervolume, in the case of 3 and more attributes-dimensions). Furthermore, we require that equal Severity values should trigger the same adaptation actions. In the case of two attributes, finding an analytical expression to determine the splitting of a square area zone to three equal zones, also satisfying the requirement for equal Severity values (as seen in Fig. 1), is a difficult but nevertheless achievable task. However, as the number of dimensions increases to three or more, the problem becomes greatly exaggerated. This means that a solution based on an alternative mathematical principle should be found.

Table 1 Severity zones bounds

Severity Zone	Sub-interval lower bound	Sub-interval upper bound
1	0	$\frac{\sqrt{n}}{m}$
2	$\frac{\sqrt{n}}{m}$	$\frac{2\sqrt{n}}{m}$
3	$\frac{2\sqrt{n}}{m}$	$\frac{3\sqrt{n}}{m}$
…		
m	$\frac{(m-1)\sqrt{n}}{m}$	$\sqrt{n}$

Such a solution is possible, if we consider the use of a random simulation. For this purpose, we simulate the normalized monitoring attribute values for situations having k normalized attributes, by retrieving random points $\mathbf{s}_i$ from the Cartesian product of possible normalized values of each of the k attributes. Since normalization converts the values of attributes to percentages, we require that $\mathbf{s}_i \in [0,1]$x$[0,1]$ x … x$[0,1]$. Each point $\mathbf{s}_i$ reflects the values of the monitoring attributes of a possible detected situation. The choice of each $\mathbf{s}_i$ is uniformly random, so we can assume that the number of points which should belong to each of the Severity zones will be equal, if their volume (i.e., *event space)* is equal. Thus, a number of p random points is chosen and sorted. Then, supposing that there exist m zones (areas) which should be determined, we determine the ratio $z = \lfloor p/m \rfloor$, where z is the number of points per area. Finally, we calculate the maximum Severity values for the first m-1 zones (the last Severity zone always has the value of $\sqrt{n}$ as already stated above) by calculating the Severity value of the $(i \cdot z)$ th element, where $1 \leq i \leq m - 1$. When these values are known, the Severity zone of a detected situation can be determined by comparing the Severity value calculated to the Severity values of each Severity zone.

The complex calculations outlined above, are based on a simple Markov Chain Monte Carlo simulation which is a well-established technique in the field of engineering. Although the calculation of complex integrals which is needed in this – and in more complex cases – is difficult to perform in an analytic fashion, the Markov Chain Monte Carlo simulation provides a satisfactory approximation which can readily be used.

Cloud adaptation techniques

In this Section we present different techniques based on Severity which guide application adaptation. Further and in order to highlight the novelty of our approach, we present adaptation techniques based on the commercial offerings of major cloud vendors. The latter are used as a baseline in Section 6 to help us evaluate the usefulness

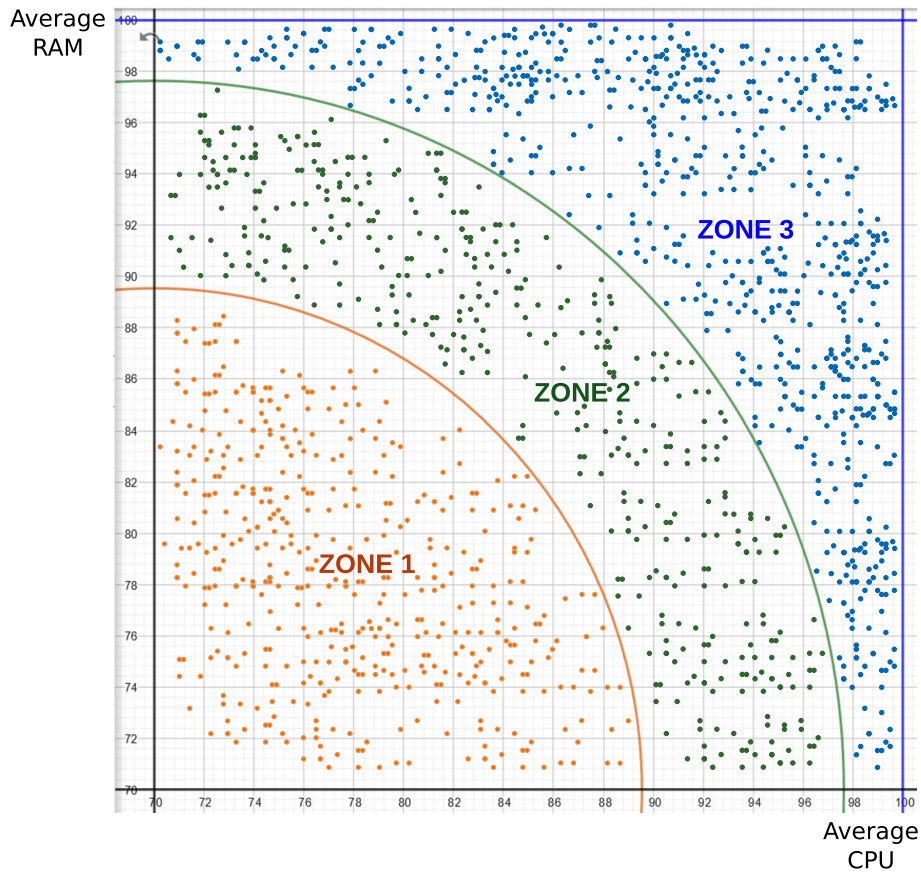

Fig. 1 The situation space of the example Elasticity rule of Listing 2, split into 3 equal Severity Zones

of Severity. Sections 4.1 and 4.2 describe two techniques which are based on commercial offerings discussed in the state-of-the-art analysis (Section 2). In Sections 4.3 through 4.9 we define seven different techniques which can govern the scaling of an application, based on Severity. Each of these techniques serves a different design goal. The efficiency of each design is evaluated in Section 7.

While designing any technique using the Severity value, we should be prepared to balance the detail of the response between small load fluctuations and the need to handle sudden workload peaks or troughs, which theoretically can be several times bigger/smaller than the current workload. Here, we discern between seven basic flavours which are based on it: Absolute severity value, Normalized absolute severity value, Normalized absolute severity control loop, Simple severity zones, Relative severity zones, Severity value and Normalized severity value. In the equations presented below, we assume that the absolute Severity value is ***as***, Severity value of a situation is ***s,*** the Severity zone of the situation is ***sz***, the Severity value of the threshold is ***ts*** and that the maximum Severity value possible is ***ms.*** The current number of instances of an application is assumed to be **ci** and the new number of instances after the adaptation is assumed to be **ni**.

With the exception of the Maximum attribute control loop technique, the adaptation instances which are determined by each technique in an adaptation action, are rounded to the nearest integer. The Maximum attribute control loop technique uses as an exception the ceiling value of the calculated number of adaptation instances.

Table 2 summarizes the design traits of the methods to be discussed in the next subsections. Each method is characterized by its origin (whether it attempts to simulate related commercial offerings) and its dynamic behaviour. If a method uses more than metric value present in an SLO rule, to establish the suggested new number of instances, a positive indication appears in the third column. Similarly, if it can spawn or deallocate a non-predefined number of instances, a positive indication appears in the third column. Finally, the fourth column provides a measure of the relative number of instances we expect the method to change in a scaling event. Methods exhibiting similar characteristics use input data (the threshold, the metric values) in a different manner, so we propose them as possible alternatives

Table 2 Characteristics of adaptation techniques

Technique	Inspired from commercial offerings	Uses individual values from multiple metrics	Dynamic resource (de)allocation	Aggressive (de) allocation of instances
Simple threshold	✓	✗	✗	Very Low
Maximum attribute control loop	✓	Partly	✓	Medium – also depends on the thresholds set
Absolute severity value	✗	✓	✓	Very High
Normalized absolute severity	✗	✓	✓	Medium
Normalized absolute severity control loop	✗	✓	✓	Low – also depends on the thresholds set
Simple severity zones	✗	✓	✗	Custom (In tests, was configured as Low)
Relative severity zones	✗	✓	✓	Custom (In tests, was configured as Low)
Severity value	✗	✓	✓	High
Normalized severity value	✗	✓	✓	Low

which can be more effective under different circumstances.

Techniques which feature higher dynamicity than others can potentially respond quite well to sudden workload changes; however, they might also respond too aggressively when a small workload change occurs, and thus be unstable. Still, techniques which have lower dynamicity, might not be efficient in handling workload spikes, but can be more stable when small adaptations are needed.

Simple threshold

The Simple threshold technique is inspired by the offerings of major cloud providers, and the THRES technique described in [14]. It adds or removes one instance for as long as the thresholds of a rule are violated. The new number of instances after an adaptation is ni = ci ± 1 (the plus sign is for a scale out rule and the minus sign for a scale-in rule). The aim of this technique is to allow fine-grained adaptations, however it cannot efficiently handle sudden workload changes.

Maximum attribute control loop

The Maximum attribute control loop technique is inspired by the offering of the Kubernetes HPA. The technique adds or removes processing instances, trying to keep a number of monitoring attributes close to their thresholds and choosing the greatest adaptation, i.e., the maximum number of instances which should be added/removed. Its dynamicity renders it is suitable for both small and greater workload changes. The attribute which triggers the greatest adaptation is referred to as the 'maximum_attribute' and its threshold as 'maximum_attribute_threshold'. The new number of instances after a scale-out or a scale-in adaptation appears in Eq. 4.

Equation 4 New instances determined for a scale out rule using the Maximum attribute control technique

$$ni = ci \cdot \frac{maximum\ attribute}{maximum\ attribute\ threshold} \tag{4}$$

Absolute severity value

The Absolute severity value technique uses the absolute Severity value from a situation, which is calculated by assuming that $t_i = 0$ in Eq. 2 and Eq. 3. In techniques using the absolute Severity value, the values of the thresholds of each metric are only used to trigger the rule, but do not affect the new number of instances. The new number of instances after an adaptation using this technique is ni = ci(1 ± ***as***)

As the maximum possible value of Severity increases linearly with the number of attributes which are involved in a situation, this technique is oriented to handle sudden spikes which are caused by a precise combination of multiple metrics. However, in the case of smaller workload fluctuations it can introduce unnecessarily large reconfigurations. As mentioned above, ***as*** reflects the absolute Severity value.

Normalized absolute severity

The Normalized absolute severity value technique tries to stabilize the instances of the application using the normalized absolute Severity value – which is obtained by dividing the absolute Severity value with the maximum possible Severity value. This technique allows a reaction which is proportional to the actual metric values (and does not use the threshold values except for its triggering). The new number of instances after an adaptation using this technique is shown in Eq. 5.

Equation 5 New instances determined for a scale out (plus sign) and a scale in (minus sign) rule using the Normalized absolute severity technique

$$ni = ci\left(1 \pm \frac{as}{ms}\right) \tag{5}$$

As an example, if the absolute Severity of a scale in rule was calculated to be 1.2, the maximum Severity for this rule is 2 and the current number of instances is 10, from Eq. 5 the new number of instances will be 4, meaning that 6 of the instances will be deactivated.

Normalized absolute severity control loop

The Normalized absolute severity control loop technique tries to stabilize the normalized absolute Severity value around the Severity value of the threshold. To avoid continuous adaptations, an upper threshold and a lower threshold are used, separated by a customizable margin inside which no adaptation is triggered. This technique can be seen as a generalization of the Maximum Attribute Control Loop technique of Section 4.2 to use Severity (also not using only the maximum value). Depending on the thresholds set the technique can be very conservative (and stable) or quite liberal in its recommendations. The new number of instances after an adaptation using this technique is shown in Eq. 6:

Equation 6 New instances determined for a scale out (plus sign) or a scale in (minus sign) rule using the Normalized absolute severity control loop technique

$$ni = ci \pm ci \cdot \left(\frac{as}{ts} - 1\right) \tag{6}$$

where the ***ts*** value corresponds to either the upper or the lower threshold for a scale out and a scale in rule, respectively.

As an example, if the absolute Severity of a scale in rule was calculated to be 0.9, the lower threshold Severity for this rule is 0.6 and the current number of instances is 10, from Eq. 6 the new number of instances will be 5, meaning that 5 instances will be deactivated.

Simple severity zones

The Simple severity zones technique uses the concept of Severity zones to find the number of instances which should be added to the infrastructure which is currently used. The flavour of the technique which was tested in our experiments used 3 Severity zones, which resulted in 1, 2 or 3 instances being added or removed from the current infrastructure as appropriate. This technique can be viewed as a somewhat more aggressive generalization of the Simple threshold technique of Section 4.1, using Severity. As the number of instances it can spawn or deallocate are constant, this method is ideal for situations in which the application platform is stable and the workload is only gradually modified. The new number of instances after an adaptation using this technique is shown in Eq. 7:

Equation 7 New instances determined for a scale out (plus sign) or a scale in (minus sign) rule using the Simple severity zones technique

$$ni = ci \pm sz \tag{7}$$

As an example, if the processing infrastructure currently has 4 instances and the Severity zone of a situation triggered by a scale out rule was found to be 2, from Eq. 7 the new number of instances will be 4 + 2 = 6.

Relative severity zones

The Relative severity zones technique uses the concept of Severity zones to add (or remove) a percentage of the current number of instances to the processing infrastructure. Hence it is more dynamic (in general) than the Simple severity zones technique described in Section 4.6. Higher values of the ***k*** constant indicate more pronounced adaptations. It is recommended that $k \cdot \max(sz) \leq 1$, otherwise in extreme scale-in adaptations all available instances will be deactivated (if no other specific handling of this issue occurs). The flavour of the technique which was tested used 3 Severity zones, and k = 0.1. The updated number of instances based on this technique is shown in Eq. 8:

Equation 8 New instances determined for a scale out (plus sign) or a scale in (minus sign) rule using the Relative severity zones technique

$$ni = ci\left(1 \pm k \cdot sz\right) \tag{8}$$

To illustrate, let us consider that a scale out rule has been triggered and the resulting situation is in the second Severity zone while k = 0.25. Then, if the current number of instances is 4, two instances will be removed bringing the total number of instances to 2.

Severity value

The Severity value technique uses the value of the Severity which is calculated using the value of each metric threshold as t_i in Eq. 2 and Eq. 3. It is suitable both for small and large workload changes. In the case though that the thresholds are not tuned to the workload, it can be unstable and introduce a large number of reconfigurations. The updated number of instances based on this technique is shown in Eq. 9:

Equation 9 New instances determined for a scale out (plus sign) or scale in rule using the Severity value technique

$$ni = ci\left(1 \pm s\right) \tag{9}$$

For example, if a scale in rule has been triggered and its Severity is 0.5 while the current number of instances

is 4, two instances will be removed bringing the total number of instances to 2.

Normalized severity value

The Normalized severity value technique uses the value of Severity calculated as in the case of the Severity value technique, divided by the maximum Severity possible to obtain a normalized (and smaller overall) result. This technique can be used to obtain more conservative adaptation results when the thresholds are not known to be tuned to the workloads, and also on workloads of smaller variability. The new number of instances after the application of this technique appears in Eq. 10, for scale out and scale in rules respectively.

Equation 10 New instances determined for a scale out (plus sign) or scale in (minus sign) rule using the Severity value technique

$$ni = ci\left(1 \pm \frac{s}{ms}\right) \tag{10}$$

To illustrate, if the Severity value of a scale out rule was found to be 1.0, the maximum Severity value is 2.0 and the current number of instances is 10, the new number of instances will be 15, meaning that 5 new instances will be added.

Illustrative scenario implementation

In this section, we will provide a walkthrough of the situation detection process, from the monitoring data published by instances comprising the cloud application to the adaptation which is decided. We will focus on the calculation of the Severity value, and how this can be translated to an adaptation using the Simple severity zones technique.

Let us assume that the elasticity rule illustrated in Listing 2 (repeated here for convenience) is used to process incoming events:

Listing 3 Elasticity Rule example

For fragid = *VideoTranscoder*
if $AverageCPU_{cluster}$>70% ***and*** $AverageRAM_{cluster}$>70%
within Timewindow=10 *minutes*
and 30 *minutes* ***have passed from previous adaptations then*** *Scale_out*

The above rule states that if the average CPU and memory usage on all devices hosting the component 'VideoTranscoder', surpasses 70% in a time window of 10 min, a new scale out adaptation action decision should be issued – provided that no previous adaptation event has occurred in the last 30 min (cooldown period).

Situation detection

We consider that at a certain time point, a new observation is detected, indicating that over the last 10 min, the average values for CPU and RAM were 92% and 71% respectively. Moreover, no adaptation event has occurred in the last 30 min. As the average CPU and RAM values are trespassing both thresholds which have been set to 70%, the rule will be triggered, and a situation will be detected. To calculate the Severity of the situation, the detected values should be normalized:

$$Normalized(CPU_1) = \frac{abs(92–70)}{abs(100–70)} = \frac{22}{30} \sim 73.3\%$$

$$Normalized(RAM_1) = \frac{abs(71–70)}{abs(100–70)} = \frac{1}{30} \sim 3.3\%$$

In some of the techniques which were presented in Section 4, the absolute Severity value is used. Had we considered such a technique, the detected attribute values would not be normalized, and the original metric values would be used instead.

Once the final metric values to be used are known, the Severity of the situation can be calculated:

Listing 4 The calculated Severity of the situation

$$Severity(CPU, RAM) = \sqrt{1 \cdot 0.733^2 + 1 \cdot 0.033^2} = 0.806$$

Using severity zones – based techniques

To accurately find the Severity zone for a detected situation, we need to know the number of the Severity zones which will be used, and the number of attributes which are monitored in each situation. To satisfy the first need, we assume throughout this work that three Severity zones will be used. To satisfy the second need, we can see from the active rule (Listing 3), that there are two attributes which are monitored ($AverageCPU_{cluster}$ and $AverageRAM_{cluster}$). Following the random point generation and sorting, the Severity zones are determined to have the upper bounds indicated in Table 3.

As already stated, the upper bound of each rule refers to the deviation from the threshold values, which when normalized are equal to zero. The highest upper bound of Severity zone 3 reflects the situation which has monitoring attribute values with the maximum deviation from the thresholds set by the DevOps observed when all monitoring attributes reach their maximum value, and is equal to $\sqrt{n}$ – in this case $\sqrt{2}$.

The Severity of the situation calculated in Listing 4 is greater than the first upper bound, and as a result the situation is marked as Severity zone 2. This can be visualized in the following illustration, depicting the Severity classification of all points which indicate a detected situation. The reported average RAM consumption is indicated in the vertical axis, while the reported average CPU consumption is indicated in the horizontal axis. Following the upper bound calculation method

Table 3 Upper bounds for three Severity zones with two attributes

Severity zone	Calculated upper bound
1	0.651
2	0.921
3	1.414

presented above, the three Severity zones are depicted in Fig. 2.

As expected, the situation initially observed (average CPU = 92%, average RAM = 71%) is located inside the second Severity zone.

In the case of a more complex rule with three attributes and four Severity zones, situations would appear as points in a three-dimensional space, and three spherical shells would be needed to mark the boundaries of the zones. In Fig. 3, red points indicate possible situations, and the three shells indicate the limits of each Severity zone. Situations which are 'outside' all shells belong to zone 4, while situations which are 'inside' all shells belong to zone 1.

Prototype implementation

Apart from developing a simulator to rapidly assess the response of a system using any of the techniques discussed in Section 4 we also developed an application adaptation manager – implementing the functionality of the Situation Detection Mechanism (SDM), and also incorporating the calculation of the Simple severity zones technique. The operation of this software with real monitoring data, proves the feasibility of our approach. The source code of our software is publicly available in Gitlab: https://gitlab.com/prestocloud-project/situation-detection-mechanism-v2

An overview of the subcomponents involved in the process of the situation detection and the subsequent platform adaptation is provided in Fig. 4.

The Situation Detection Mechanism is assumed to be part of a platform which manages the adaptation of a cloud application by issuing appropriate scaling directives. Therefore, the subcomponents illustrated in Fig. 4 are not part of the cloud application, but instead form the internal architecture of the SDM. The architecture of the Situation Detection Mechanism is structured around the usage of a common message bus, in this case the RabbitMQ[1] broker. This allows not only the decoupling of subcomponents, but also an abstraction layer over the monitoring data which is sent by monitored devices. The mechanism which is used to retrieve monitoring data from the application is agnostic to the Situation Detection Mechanism. The SDM can handle the monitoring of a processing infrastructure composed of any kind of processing machines (VMs, Physical Machines PMs, Containers, Network and edge devices), and receiving any number of processing attributes values, with the proper configuration by the DevOps. Information on the situations which are detected by the SDM is sent through the message bus to an external component, the Resources Adaptation Recommender (RARecom), which issues the actual scaling directives.

The input from the DevOps which triggers the monitoring cycle are the elasticity Rules which are created using an appropriate user interface (UI). This user interface allows the DevOps to select the monitoring metric(s) which should be monitored for a particular component of the deployed cloud application, and the threshold(s) which need to be set. Once an elasticity rule is received by the SDM, it subscribes to the RabbitMQ broker which is directly connected with the infrastructure instances. Then, it can start consuming monitoring events which are related to the metrics of the rule. The flow of monitoring events appears over the message bus in the uppermost part of the figure.

The *Rule Interpreter* module undertakes the conversion of elasticity rules to queries understandable by the Siddhi streaming input processor [34]. These queries use monitoring data to detect a new situation. To translate an elasticity rule, we first determine the monitoring metrics which are specified in it. Then the source (Siddhi stream) for these values is determined, and a suitable Siddhi query is programmatically constructed to retrieve the values. In Listing 2 above, we described an example elasticity rule which can be used to govern the scaling of an application outwards. The representation of the generated Siddhi query for this rule can be seen in Listing 5. We assume that cpu_perc, mem_perc, fragid and res_inst are monitoring attributes contained in a Siddhi monitoring stream named 'serverStream' (fragid is the component id and res_inst is a unique identifier of the monitored resource).

Listing 5 Siddhi query created for example elasticity rule

```
define stream scalability_rule_stream (avg_cpu_perc double, avg_
    mem_perc double, fragid string, res_inst string);

from serverStream#window.timeBatch(600 s) [fragid == '1cc5Fragment']

select avg. (cpu_perc) as avg_cpu_perc, avg. (mem_perc) as avg_
mem_perc, fragid, res_inst

having avg_cpu_perc >70 and avg_mem_perc > 70

insert into scalability_rule_stream;
```

The check for a possible previous adaptation which could have happened within the cooldown period is implemented by Java code which is external to Siddhi.

[1] https://www.rabbitmq.com/

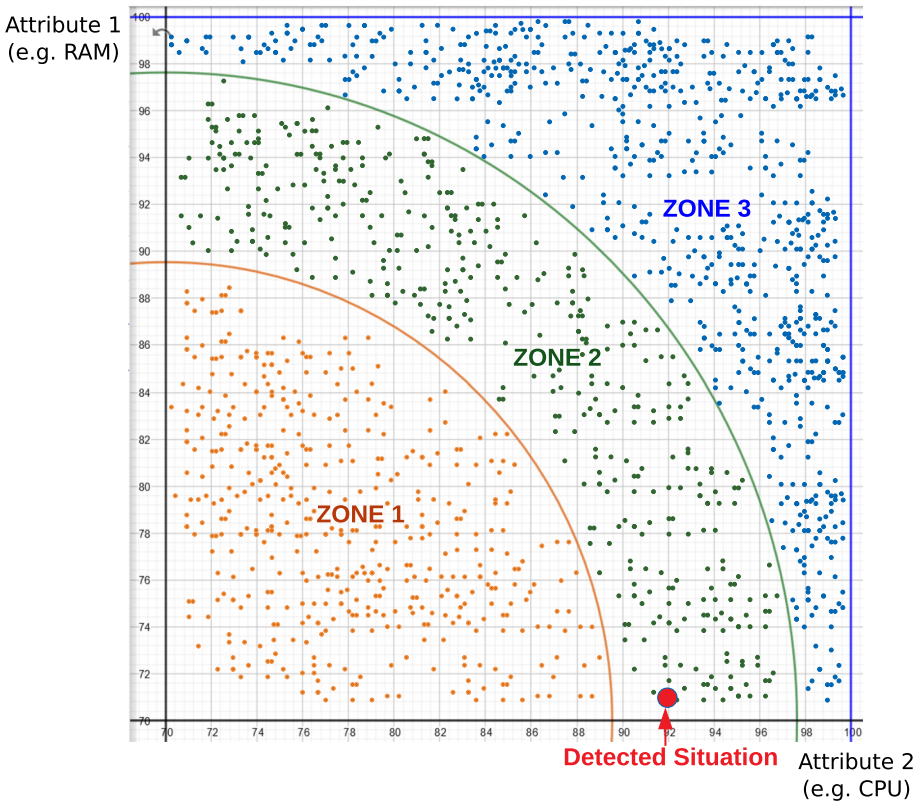

Fig. 2 Severity zones in the case of two metrics and three zones

The choice to use Siddhi builds upon our previous work [35]. Using Siddhi, the SDM can detect when the monitoring attribute thresholds set in a rule are surpassed in which case the elasticity rule is violated, and a situation is detected. Once a situation has been detected it is associated with a Severity zone by the *Severity Zone Calculation* module which performs the calculations which are required. Subsequently, the detected situation and information related with its Severity is published to the broker using an appropriate situation event for further processing by the *RARecom* which will consume it. The RARecom can then process these events to determine the number of instances that will be needed for the scaling adaptation (according to the flavour of the Severity zones technique which is desired, or even perform a new and independent assessment of the situation using another technique).

Evaluation

Benchmark & error metric choice

In order to evaluate the efficiency of our approach, we conducted a series of experiments – simulating both components which are involved in the implementation of an adaptation, the SDM and the RARecom. Our simulations varied both regarding the input of the DevOps (i.e., the elasticity rules), and the characteristics of the incoming workload. Moreover, we experimented with a varying amount of resource provisioning time delay that follows a realistic adaptation implementation behaviour. Moreover, one of these delays was set to zero which reflects one of the key elasticity aspects of an ideal cloud platform [36].

The rule approaches which have been adopted by some of the main commercial Cloud vendors are based on the use of one or more static rules, which should be created by the DevOps [14]. Using our approach, the DevOps is required to setup only a single rule for scale out, and a single rule for scaling in (per metric combination). Then, this rule can be used as an input for a multitude of techniques based on the concept of Severity. In this section we evaluate the techniques which were discussed in Section 4 along with common approaches which are found in commercial systems – using either Simple threshold (ST), or the Maximum attribute control loop (MACL). We only use one rule for scaling up and one rule for scaling down (using 2 attributes in the case of 2-metric workloads, 3 attributes in the case of 3-metric workloads and 4 attributes in the case of 4-metric workloads). Specifically, the threshold pairs which were tested appear in Table 4.

The maximum number of attribute values which were examined simultaneously was 4, although the

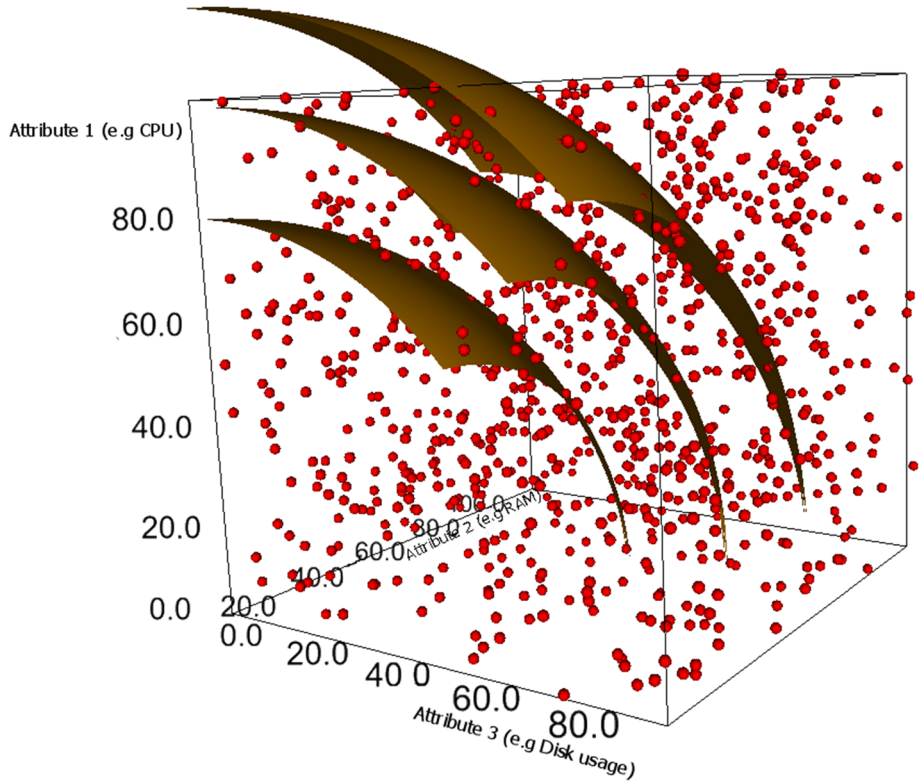

Fig. 3 Severity zones in the case of three metrics and four zones

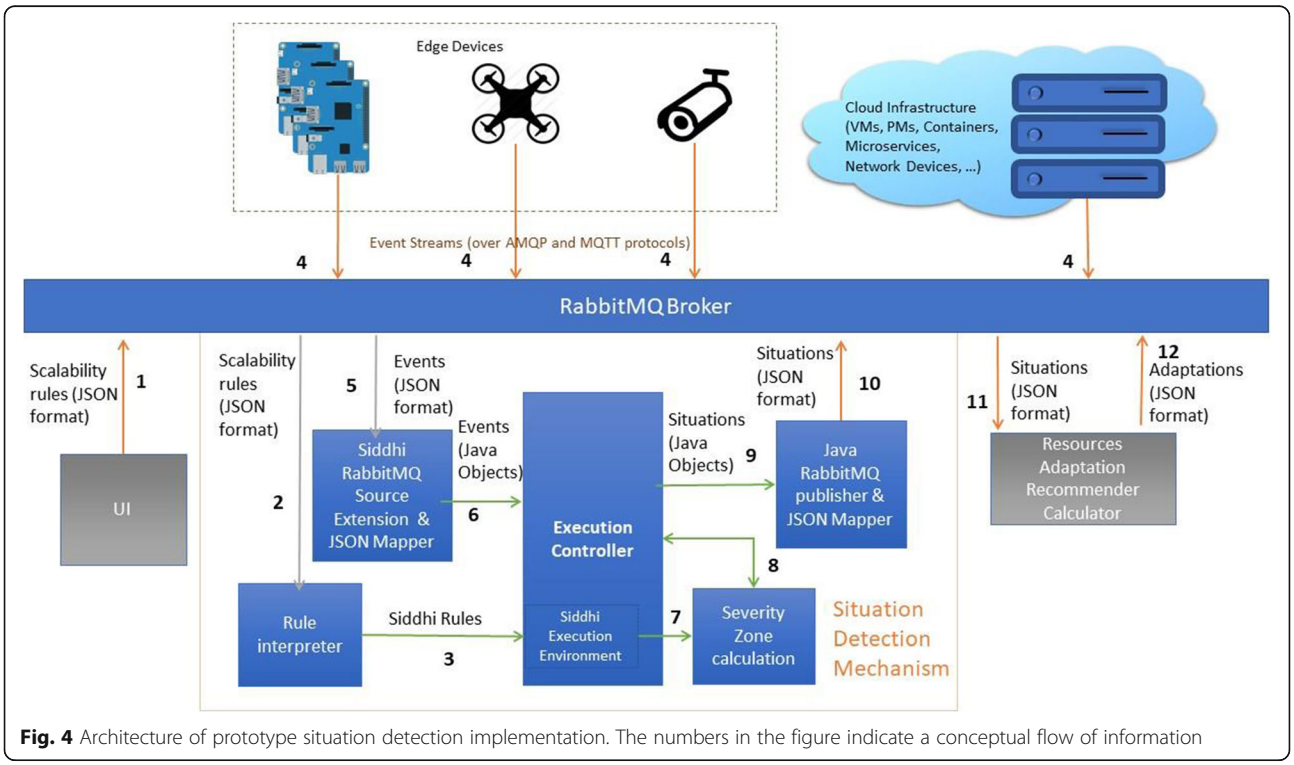

Fig. 4 Architecture of prototype situation detection implementation. The numbers in the figure indicate a conceptual flow of information

concept of Severity can handle an arbitrary number of metrics. Also, the upper threshold for both attributes was chosen to be the same for both attributes, for reasons of simplicity. At this point we assume that the nature of the workload is only roughly known to the DevOps – which in turn does not allow the fine-tuning of the rules and permits only a simple selection of the thresholds. Attributes 1 to 4 may reflect any metric (e.g., CPU, RAM, Disk usage, Network bandwidth utilization etc.). The starting point for these experiments is the definition of elasticity rules by the DevOps. These rules dictate the scaling in or scaling out of the platform – also referred to as an adaptation action – to accommodate the load which is induced by the cloud application. We then applied each pair of rules in Table 4 to handle the workloads presented in Figs. 5, 6, 7 and 8, reconfiguring the application and collecting metrics on its performance. The time window of the rules was set equal to one-fifth of the VM spawn duration with a minimum duration of 3 s (i.e., 3, 3, 6 and 12 s for the 0, 15, 30 and 60 s spawn intervals which were tested). No distinction was made between VM spawn delays and VM deallocation delays, thus whenever in this work VM spawn delays are mentioned VM deallocation delays should also be considered equal. A cooldown period of 10 s was required between successive adaptations for all rules.

Optimally, we would prefer to use the available resources to their maximum capacity, while also serving the traffic appropriately and having maximum stability. However, to attain this ideal goal, it is required to be able to accurately know the current and future demand of the service so, unavoidably some deviation (error) will exist in at least one of the above-mentioned goals. Thus, the thresholds of the rules should be created by a field expert considered able to balance the risk of service unavailability, with the number of resources which are overprovisioned and application stability. To evaluate the proposed techniques based on Severity, we have considered various benchmarking metrics which are described in Table 5.

Other approaches have also been using similar metrics in their experiments. For example [14, 18] examined the number of containers and VMs respectively (which can be related to cost) and the response time for different techniques, while [37] examined cost and execution time (which can be converted to a question between cost and availability). In a thorough review of cloud elasticity [36], the authors mention eagerness, sensitivity and plasticity (related to our 'rigidness' metric), quality of service (indirectly related to our 'availability' metric), cost, oscillatory behaviour/thrashing (related to our 'adaptations' metric) and precision (related to our 'overprovisioning' metric) as principal aspects of service elasticity which should be considered in an elastic system.

In our experiments, we used four workload types, most of which were characterized by their frequent and abrupt changes in the workload. Each workload type included two, three or four workload patterns, one for each of the metrics. We used a separate pattern per processing metric in three workloads (the green line reflects values of the first metric, while the red line reflects values of the second metric, the blue line the values of the third metric and the yellow line the fourth metric) - the linear workload used the same pattern for all metrics. In experiments using two metrics we used the red and green lines, in experiments using three metrics we used the red, green and blue lines, and in experiments with four metrics we used all four lines. Unlike other approaches (e.g., [14]) we normalized workload values against the processing capacity of a single VM, which is assumed to be 100% per processing metric resource. The x-axis of all workload types represents the time in seconds which has elapsed since the start of the experiment.

It can be readily observed that the four workload types have values which surpass 100% for each of the attributes which are involved. We interpret these values as a need for more resources which would require additional VMs. Specifically, a value pair of (1%, 200%) in workloads having two attributes means that at least two VMs are needed for the handling of this workload

Table 4 The upper and lower thresholds of the elasticity rules which were used

ID	Scale Out Rule Threshold (greater than operator)				Scale In Rule Threshold (less than operator)			
	Attribute 1	Attribute 2	Attribute 3	Attribute 4	Attribute 1	Attribute 2	Attribute 3	Attribute 4
1	70	70	70	70	30	30	30	30
2	65	65	65	65	55	55	55	55
3	80	80	80	80	70	70	70	70
4	80	80	80	80	55	55	55	55
5	90	90	90	90	80	80	80	80
6	90	90	90	90	10	10	10	10

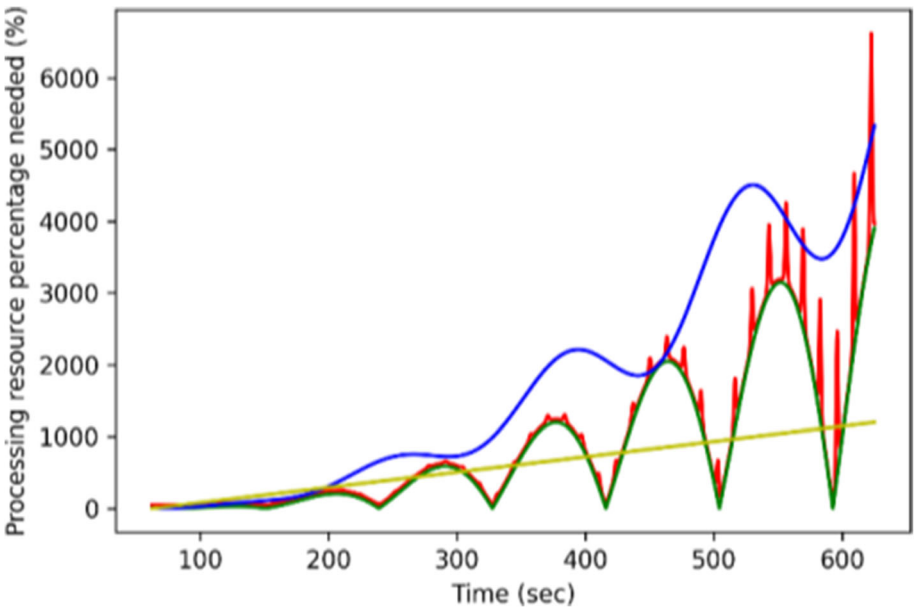

Fig. 5 Periodically increasing workload, with spikes.

type even though the first attribute only consumes 1% of the resources. In this work, we consider all VMs to have the same processing capacity, i.e., we do not take into account different VM types. Moreover, the processing capacity of a VM does not correspond to the processing capacity which is offered by a particular VM flavour of a cloud vendor.

Evaluation results

In this section, we discuss the results of each proposed technique when applied in different representative workloads, with respect to the benchmarking metrics discussed in Section 7.1. The results were gathered by means of a Python simulator which was developed specifically for the needs of this work. The simulator

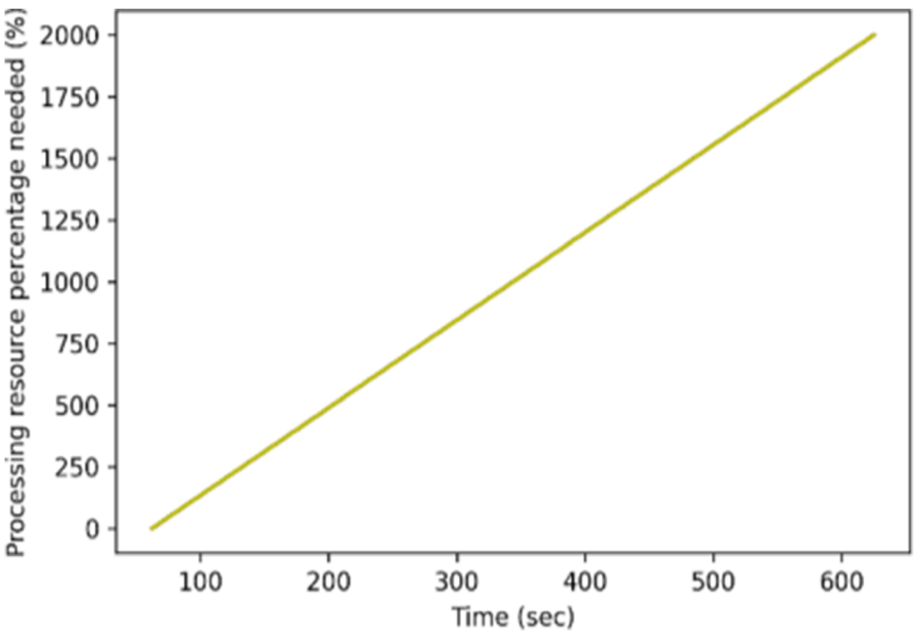

Fig. 6 Linearly increasing workload. Due to overlapping of the values, only the yellow line is visible

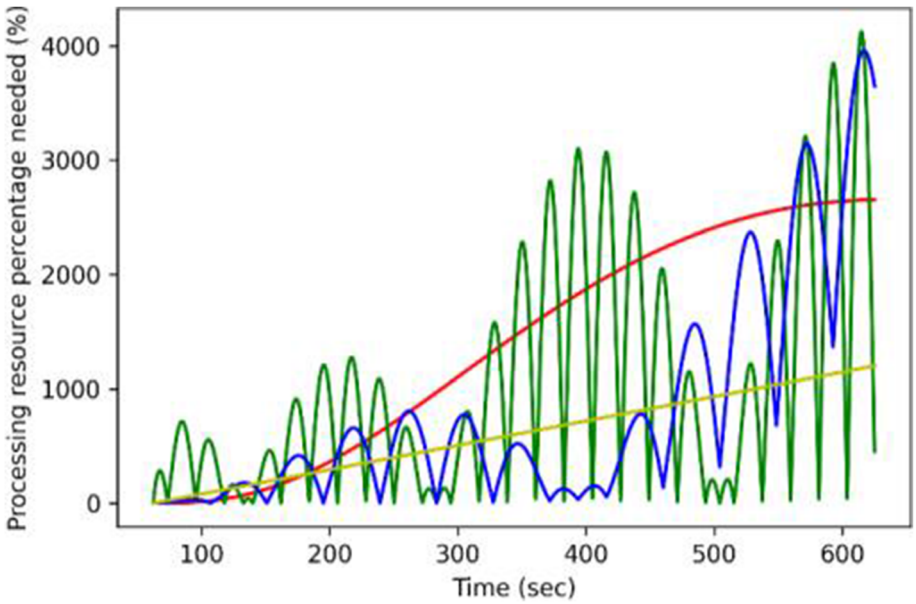

Fig. 7 Periodically increasing workload, with fluctuations

produced an output file containing the simulated timestamp - a constant time interval was added between successive timestamps, and information on the current real load on the application, the optimal number of instances which could handle the workload, and whether the workload is over the operational thresholds set by the DevOps. Both the code as well as all of the workload traces which were used are available upon request.

In Table 6, we present the minimum and maximum value for each combination of benchmarking metric, VM spawn delay and 4-metric workload, regarding the Maximum attribute control loop and Severity value

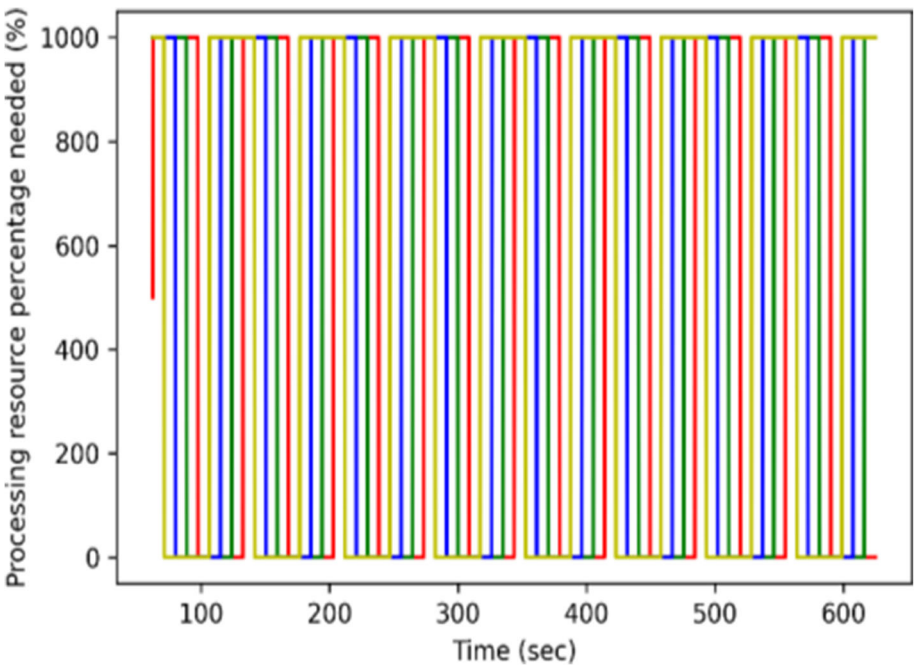

Fig. 8 Polarized workload

Table 5 Description of the benchmarking metrics used for the evaluation of Severity-based techniques

Benchmarking metric	Description
Availability	The percentage of time for which the workload pattern for a particular metric was not demanding more than the processing capacity of the infrastructure offered. We do not distinguish whether the processing capacity was exceeded by a small or large margin – in both cases the service is considered unavailable for the purpose of our experiments. Moreover, we considered that the lack of availability of the service at a particular instance of time does not influence the ability of the service to handle the workload correctly as soon as it receives the resources which are required.
Overprovisioning	The product of the extraneous VM instances which were used (compared to the optimal) with the percentage of simulation time for which they were spawned.
Rigidness	The time percentage of a simulation, for which the application was working either above or even below the rule thresholds set. For example, if a rule on a metric states that a scale out action should happen when the value of the metric surpasses 70%, while a scale in action should happen when the value of the metric drops below 30%, the system is considered to be exhibiting 'rigidness' when the value of the metric is greater than 70% or lower than 30%.
Number of scaling adaptations	The total number of scaling adaptations (associated with an addition or removal of a number of VMs) which were performed by the platform. The first deployment is also counted in this number.

techniques. The Severity value technique exhibited the best performance overall in the four criteria which were defined. We also included data for the Maximum attribute control loop technique as it had better performance than the Simple threshold technique. Therefore, it represents the best of the two techniques which are inspired from commercial offerings and are considered in this work.

Ranges were calculated from the output of the simulations which used the rule pairs which are included in Table 4. The complete dataset with the exact performance of all techniques on every rule pair, every spawn delay and every workload examined is available as part of this work [38]. All values in Table 6 reflect percentages, except for the number of scaling adaptations which is an integer. Positive values in availability reflect the percentage of the time that the platform could satisfy the load with the existing resources. Positive values of rigidness indicate the percentage of the time that the metric values of the platform were outside the acceptable range set by the greater-than and less-than thresholds. Positive values in overprovisioning indicate a percentage of superfluous VMs which were commissioned by a technique to handle the load, compared to the number of VMs which would exactly match the workload, utilizing their resources up to 100%. As discussed above this is an unrealistic case, as the DevOps needs to set thresholds

Table 6 Techniques comparison matrix (4 metric workloads)

Technique	Workload	Overprovisioning				Availability				Rigidness				# of Adaptations			
		0-delay	*15-delay*	*30-delay*	*60-delay*	*0-delay*	*15-delay*	*30-delay*	*60-delay*	*0-delay*	*15-delay*	*30-delay*	*60-delay*	*0-delay*	*15-delay*	*30-delay*	*60-delay*
Severity value	Periodically increasing with spikes	-46.2	-50.5	-43.4	-61.2	19.0	25.0	23.2	20.0	3.9	17.9	33.3	24.2	16	16	27	15
		-9.1	15.9	31.8	9.3	47.1	51.8	52.6	41.2	50.3	75.3	75.5	68.2	63	48	46	38
	Linearly increasing	10.1	0.9	22.5	19.8	61.2	46.6	47.2	41.6	0.2	13.4	13.4	15.3	23	11	11	11
		49.9	42.5	128.1	92.8	100.0	98.9	95.9	86.7	82.1	90.5	92.3	93.1	118	65	52	46
	Periodically increasing with fluctuations	-9.4	-24.3	-36.4	-30.5	30.3	28.7	25.3	28.4	0.2	5.6	10.4	15.4	8	8	9	12
		28.6	45.0	57.7	14.7	73.0	72.3	66.1	57.4	22.7	50.5	65.7	71.2	43	43	42	31
	Polarized	-61.9	-75.8	- 79.2	-48.2	30.6	12.5	12.5	28.1	29.4	54.1	63.5	33.1	32	32	38	27
		-57.8	-74.0	-72.9	-23.5	32.9	12.5	12.5	46.4	36.3	57.5	74.3	52.9	33	34	40	39
Maximum attribute control loop	Periodically increasing with spikes	-73.6	-75.4	-72.0	-65.3	9.2	9.2	8.3	9.9	33.5	39.9	34.7	36.8	31	30	27	25
		-34.9	-42.9	-36.3	-18.3	33.0	20.4	24.4	24.6	65.0	75.5	74.2	70.0	47	48	46	43
	Linearly increasing	10.1	8.1	8.8	-16.3	100.0	96.6	61.7	46.0	0.2	22.2	27.7	30.1	23	23	21	21
		50.5	48.3	43.5	66.0	100.0	99.1	95.9	87.7	12.1	40.0	89.8	91.5	31	31	52	45
	Periodically increasing with fluctuations	-11.6	-13.2	-35.7	-38.7	34.7	32.7	14.8	23.1	3.1	12.3	15.3	24.6	12	13	12	20
		31.0	26.3	26.2	40.6	73.1	66.5	63.2	60.3	10.5	24.1	40.8	55.0	25	23	32	30
	Polarized	-79.7	-79.7	-79.2	-82.2	12.5	12.5	12.5	12.5	34.9	54.1	63.5	30.1	32	32	38	31
		-70.8	-76.5	-77.4	-44.8	12.5	12.5	12.5	29.4	36.7	57.5	74.3	51.3	33	34	40	51

below 100%, but it is nevertheless the optimal case concerning overprovisioning. Negative values in overprovisioning portray a usage of a smaller number of VMs than the optimal, which by definition impacts availability.

For brevity, for the cases of 2 metric and 3 metric workloads we provide a quick summary of the results and two indicative figures. Similar to Table 6, we provide the minimum and maximum values for each of the benchmarking metrics – the top edge of each bar reflects the maximum value and the lower edge the minimum value. Percentage values are used for the measurement of the benchmarking metrics with the exception of the number of scaling adaptations metric which is an integer. Figure 9 illustrates an example case comparing the performance of the SV algorithm with the MACL algorithm, in a 3-metric workload setting. Similarly, Fig. 10 illustrates a comparison of the performance of the SSZ algorithm vs the MACL algorithm in an example case using a 2-metric workload.

Figure 9 illustrates an extreme case in which the severity technique completely outperforms the maximum attribute control loop technique. It is reminded that the range of observed values for each attribute of each technique is the result of experimenting with multiple scale-in and scale-out threshold pairs. The severity technique both allows a wider range of choices and can also obtain the best values in each of the four measured criteria. The prioritization of these criteria should be performed by the DevOps, who can then choose the most appropriate scaling technique among the suggested techniques. In this particular example, using the severity technique one may choose whether to avoid the underprovisioning of the service to obtain higher availability, lower rigidness and a lower number of adaptations. Such a choice is not available when using the maximum attribute control loop technique.

Figure 10 illustrates a case typical of the relative performance of the simple severity zones and maximum

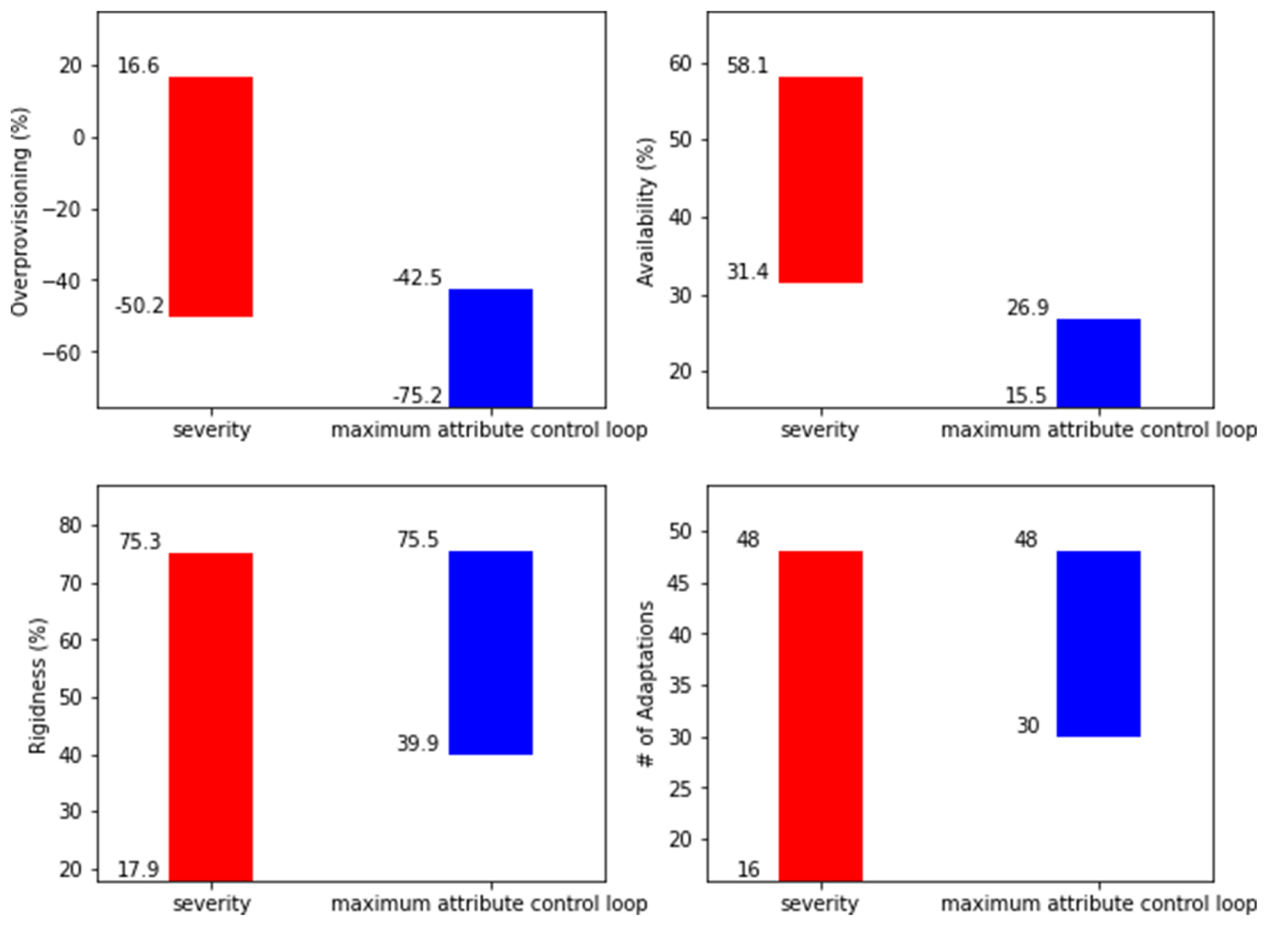

Fig. 9 An example case (2 metrics workload, 15 s delay and 'Periodically increasing with spikes' workload) in which the severity technique outperforms the maximum attribute control loop technique

attribute control loop techniques in 2 metric workloads. In this case the maximum attribute control loop technique slightly outperforms the simple severity zones technique. However, based on the experimental data the simple severity zones technique can offer better availability and smaller overprovisioning, provided that a much greater number of reconfigurations and increased amount of rigidness can be tolerated by the application.

Table 6 as well as the data collected during the experiments with 2 metric and 3 metric workloads illustrate numerous cases in which Severity-based techniques obtain better results than other techniques which are currently used by cloud vendors. To further illustrate a comparison between the different techniques, we assumed a use-case which emphasizes availability (**av**), low overprovisioning (**op**), low rigidness (**rg**), and a low number of scaling adaptations in turn (**ad**). The coefficients pertaining to each of the evaluation criteria appearing in the utility function of the particular use-case were chosen in order to assign more weight to high availability and low overprovisioning than low rigidness and a low number of scaling adaptations. The utility function using the benchmarking metrics, as well as their weights appear in bold in Eq. 11. Apart from the weights, the constant terms of the utility function are chosen to normalize the values of the evaluation criteria. Higher utility values indicate a more preferable performance.

Equation 11 Example Utility Function

$$U = \frac{\mathbf{3}*\boldsymbol{av} + \mathbf{2}*\left(\frac{1-\boldsymbol{op}}{2}\right) + \mathbf{1}*(1-\boldsymbol{rg}) + \mathbf{1}*(1-(\boldsymbol{ad}-5)*0.00877)}{7} \tag{11}$$

Tables 7, 8 and 9 illustrate the performance of each technique against each of the four workloads which were used, in the experiments conducted using 2, 3 and 4 metrics respectively. Each cell contains four values which from top to bottom reflect Utility values with a varying delay of 0,15,30 and 60 s, respectively. The following abbreviations are used: Absolute severity value (ASV); Maximum attribute control loop (MACL);

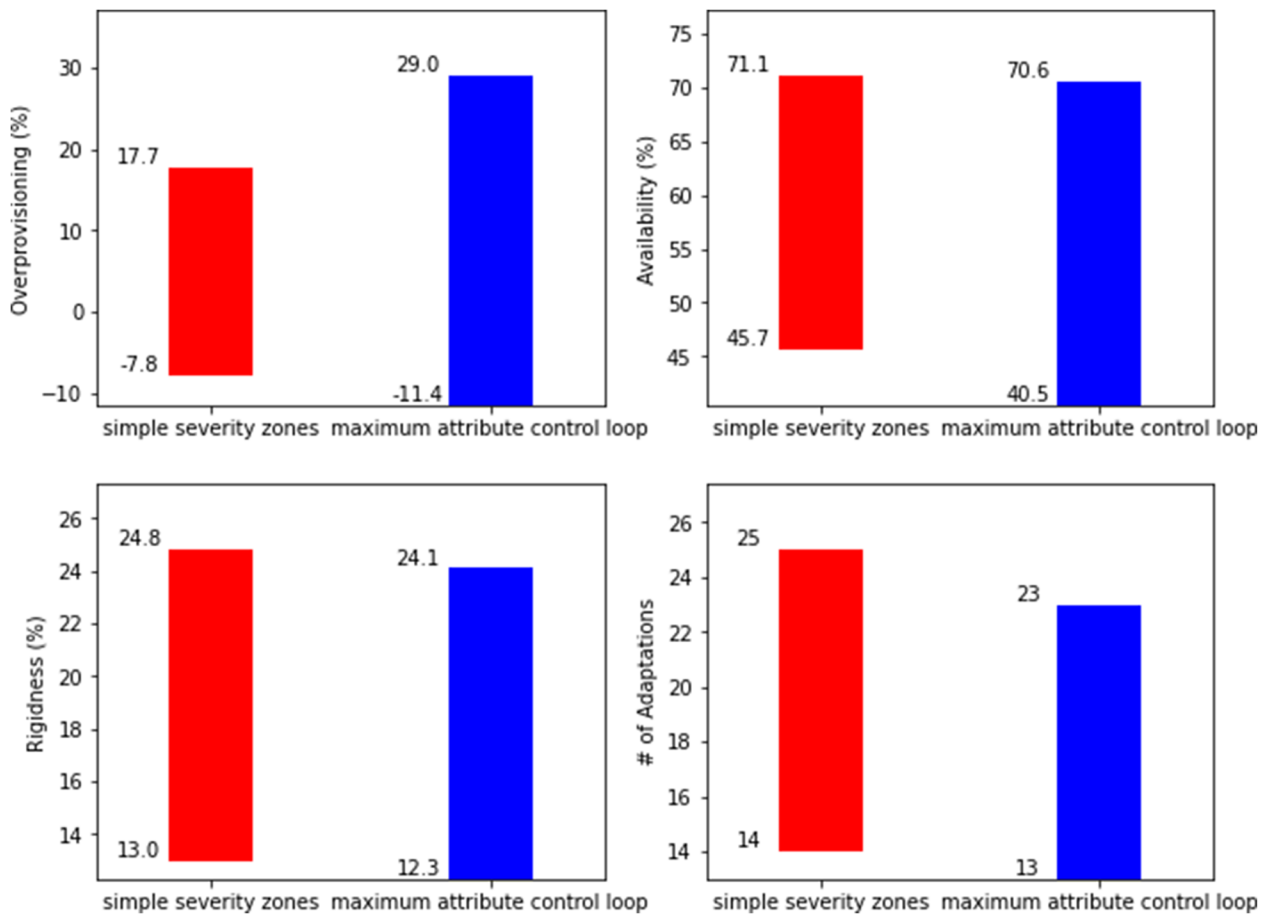

Fig. 10 An example case (3 metrics workload, 15 s delay and 'Periodically increasing with fluctuations' workload) in which the maximum attribute control loop technique outperforms the simple severity zones technique

Normalized absolute severity (NAS); Normalized absolute severity control loop (NASCL); Normalized severity value (NSV), Severity value (SV), Simple severity zones (SSZ), Relative severity zones (RSZ); and Simple threshold (ST). Underlined values indicate an (approximate) tie between algorithms, while bold values indicate the superiority of an algorithm.

From the data which is presented in Tables 7, 8 and 9, we observe that Severity-based techniques obtain the best results in the majority of the test-cases, whether 2,3 or 4 dimensions are used. The number of metrics used does not significantly influence the utility values which are obtained, although when fewer metrics are used the utility values are in general slightly increased. Also, we can observe that the increase in the delay to spawn or deallocate instances in general results in lower utility function values for the same technique.

Further, it is apparent that some techniques are more suitable for particular cloud application settings, while others are consistently outperformed. The Simple threshold technique does not perform well under any workload, indicating that either a different number of instances should be added/removed in challenging workloads, or it should not be used. The Maximum attribute control loop technique is a better contender, but it is not so effective when the spawn delay is increased to 60 s. This can be attributed to its control-loop character which tries to stabilize the values of the metrics around a desired threshold. However as most workloads change rapidly and its decisions are enforced with a delay, the stabilization is obsolete – resulting in either overprovisioning, or underprovisioning and loss of availability. This characteristic is shared with the Normalized absolute severity control loop method. On the contrary, the Simple severity zones technique consistently attains the best results when the spawn delay is 30 and 60 s and is excellent in handling linear-like workloads. Moreover, the Severity value technique attains excellent results when the delay is zero (and in 10 of the 12 cases when the delay is 15), has consistently the best performance in the 'periodically increasing with fluctuations' workload, and is the technique which attains the best value more often.

Table 10 contains the average improvement of the utility function from the usage of Severity-based techniques.

To determine the values of Table 10 we need to determine the best Severity-based and commercially inspired techniques. The sole criterion which is used to find the best technique is the higher utility function value it attains using the most favourable thresholds (for it).

For the first row of Table 10 we calculate the improvement of the best Severity-based adaptation technique against the best commercially inspired adaptation technique per workload and VM spawn delay (16 combinations). The average of this improvement is used to determine the values of this row. To fill the values of the second row, we determine the Severity-based technique which has the highest average utility value across all workloads and VM spawn delays (for the particular workloads). We then calculate the average improvement when using the best Severity-based technique against the best of the commercially inspired techniques. The third row contains the best of the Severity-based techniques, in terms of the highest average utility value, calculated over the best choice of thresholds for each technique in each experiment setting (i.e., workload and spawn delay).

It is important to note that in all cases the highest values of the utility function are produced using Severity-based techniques. In the three experiment sets (with 2, 3 and 4 metrics), a total of 48 combinations of workloads and VM spawn delays were evaluated. The Maximum attribute control loop technique was only

Table 7 The performance of each technique using 2-dimensional workloads and a variable spawn delay

Workload	ASV	MACL	NAS	NASCL	NSV	SV	RSZ	SSZ	ST
Linearly increasing	0.78	0.82	0.78	0.82	0.80	0.82	0.82	0.82	0.82
	0.66	0.77	0.74	0.77	0.76	0.78	0.77	**0.78**	0.77
	0.42	0.74	0.55	0.74	0.69	0.69	0.74	**0.75**	0.74
	0.64	0.66	**0.69**	0.68	0.68	0.67	0.65	0.68	0.68
Periodically increasing with spikes	0.65	0.63	0.64	0.67	0.72	0.72	0.72	0.72	0.63
	0.59	0.59	0.56	0.60	0.62	0.61	0.61	**0.65**	0.60
	0.52	0.61	0.51	0.61	0.56	0.49	0.61	**0.64**	0.61
	0.57	0.61	0.59	0.61	0.60	0.58	0.62	**0.63**	0.60
Periodically increasing with fluctuations	0.70	0.69	0.69	0.69	0.69	**0.71**	0.69	0.67	0.59
	0.63	0.66	0.67	0.65	0.69	0.69	0.67	0.65	0.56
	0.54	0.65	0.67	0.63	0.60	0.67	0.63	0.66	0.56
	0.63	0.59	0.64	0.57	0.62	0.65	0.57	0.65	0.53
Polarized	0.65	0.60	0.58	0.60	0.56	**0.65**	0.54	0.57	0.54
	0.57	0.57	0.57	0.57	0.57	0.57	0.53	0.48	0.53
	0.56	0.55	0.55	0.55	0.54	0.55	0.54	0.53	0.54
	0.60	0.62	0.63	0.62	**0.64**	0.61	0.62	0.61	0.62

Table 8 The performance of each technique using 3-dimensional workloads, and a variable spawn delay

Workload	ASV	MACL	NAS	NASCL	NSV	SV	RSZ	SSZ	ST
Linearly increasing	0.78	0.82	0.78	0.82	0.80	0.82	0.82	0.82	0.82
	0.66	0.77	0.74	0.77	0.76	0.78	0.77	0.78	0.77
	0.42	0.74	0.55	0.74	0.69	0.69	0.74	**0.75**	0.74
	0.64	0.66	**0.69**	0.68	0.68	0.67	0.65	0.68	0.68
Periodically increasing with spikes	0.63	0.53	0.61	0.55	0.62	**0.66**	0.54	0.53	0.49
	0.55	0.47	0.55	0.49	0.60	**0.61**	0.52	0.54	0.48
	0.50	0.48	0.51	0.47	0.59	0.57	0.50	0.59	0.47
	0.55	0.49	0.59	0.50	**0.60**	0.57	0.54	0.58	0.50
Periodically increasing with fluctuations	0.69	0.68	0.68	0.68	0.67	**0.70**	0.68	0.66	0.57
	0.62	0.65	0.66	0.64	**0.68**	0.67	0.66	0.64	0.54
	0.53	0.63	0.65	0.62	0.59	0.65	0.62	0.64	0.53
	0.62	0.58	0.62	0.56	0.61	0.63	0.56	0.63	0.51
Polarized	0.59	0.53	0.52	0.53	0.49	0.59	0.48	0.51	0.48
	0.50	0.51	0.50	0.51	0.50	0.50	0.47	0.45	0.47
	0.50	0.49	0.48	0.49	0.48	0.48	0.48	**0.53**	0.48
	0.57	0.56	**0.60**	0.56	0.59	0.58	0.55	0.56	0.55

thrice able to equal this exact maximum value – which besides was attained by the Normalized absolute severity control loop technique. In Tables 7, 8 and 9 above this appears to happen more often due to rounding.

From the evaluation of the scaling methods, the following directions on the use of our approach can be established:

- Using data from additional metrics (even partially, as the Maximum Attribute Control Loop technique does) in general leads to better estimations. Therefore, we support the research on algorithms which use data from multiple metrics, when all of them are related to the need for scaling.
- Some "non-functional", desirable characteristics when choosing a particular algorithm are its adaptability, its extensibility and its explainability. Severity as a concept lends itself to many extensions; in addition, the algorithms which are based on it are both adaptable and explainable.
- Techniques which have a very aggressive or very mild spawn/deallocation behaviour, are not recommended for rapidly changing workloads, when threshold-based rules are to be used.
- By consulting the full results of the simulations, we can observe that even for single evaluation metrics (e.g., Overprovisioning) Severity-based techniques attain the best (in this case the lowest) value in the greater majority of test cases, independent of the number of metrics used, or the workload or the spawn delay. Moreover, in many of the cases that commercial-based scaling techniques (Simple threshold, Maximum attribute control loop) attain the best value, this value is also produced by a Severity-based technique.

Discussion

The approach which has been presented in this work can be used to improve the response of systems which face difficult workloads. It can abstract the input

Table 9 The performance of each technique using 4-dimensional workloads, and a variable spawn delay

Workload name	ASV	MACL	NAS	NASCL	NSV	SV	RSZ	SSZ	ST
Linearly increasing	0.78	0.82	0.78	0.82	0.80	0.82	0.82	0.82	0.82
	0.66	0.77	0.74	0.77	0.76	0.78	0.77	0.78	0.77
	0.42	0.74	0.55	0.74	0.69	0.69	0.74	**0.75**	0.74
	0.64	0.66	**0.69**	0.68	0.68	0.67	0.65	0.68	0.68
Periodically increasing with spikes	0.59	0.50	0.57	0.51	0.58	**0.62**	0.51	0.50	0.45
	0.52	0.45	0.52	0.46	0.58	**0.59**	0.50	0.52	0.45
	0.47	0.46	0.48	0.44	0.56	0.54	0.47	**0.57**	0.45
	0.53	0.47	0.56	0.48	**0.57**	0.54	0.51	0.55	0.47
Periodically increasing with fluctuations	0.69	0.68	0.68	0.68	0.67	**0.70**	0.68	0.66	0.56
	0.62	0.65	0.66	0.64	0.67	0.67	0.66	0.64	0.53
	0.53	0.63	0.65	0.62	0.59	0.65	0.61	0.64	0.53
	0.62	0.57	0.61	0.56	0.60	0.62	0.56	**0.63**	0.50
Polarized	0.57	0.51	0.50	0.51	0.47	0.57	0.45	0.49	0.45
	0.48	0.48	0.48	0.48	0.48	0.48	0.44	0.44	0.44
	0.47	0.46	0.46	0.46	0.45	0.46	0.45	**0.50**	0.45
	0.58	0.52	0.55	0.53	**0.60**	0.58	0.51	0.56	0.51

Table 10 Improvement of utility function values from Severity-based techniques

	2-metric workloads	3-metric workloads	4-metric workloads
Maximum improvement over best commercial technique evaluated	4.09%	7.86%	8.54%
Best single-technique improvement over best single commercial technique evaluated	1.26%	5.88%	6.53%
Best Severity-based technique	Simple severity zones	Severity value	Severity value

necessary for devising elasticity rules and help the DevOps to guide the operation of a cloud application in a more effective manner. Familiar concepts found in traditional rule-based systems, such as aggregations, thresholds and cooldown intervals are still the basic building blocks. Therefore, it also reaps the benefits associated with rule-based adaptivity, such as lower updating overhead, relative genericity with respect to the workload managed, lower computational complexity when compared to other approaches [39]. Moreover, it can automatically use information from a multitude of monitoring attributes which can be provided in each elasticity rule and not only from a limited, hard-coded selection between average CPU, response time and number of requests.

In contrast to traditional rule-based approaches, the choice of the metrics which should be monitored, and the appropriate threshold values, needs to be complemented by the choice of an appropriate scaling technique. Depending on the nature of the workload and the time required to spawn new processing instances, the DevOps should choose the technique which is the most appropriate. As an example, if only two metrics are considered, availability is a priority, the spawn delay is 60 s and the workload is similar to the periodic workload we used, a severity-based technique such as normalized absolute severity, severity value and absolute severity value are worth considering. If attaining the most stable deployment (i.e., having low rigidness and a low number of adaptations) in 3 metrics periodically increasing workloads is essential, then the severity value and normalized severity techniques can be considered. Overall, the workloads which we provide and the data associated to each of them, can serve as an initial point of reference for further work and investigation.

The value of our proposal was demonstrated using a utility function which prioritizes the correct provisioning and service availability while also favouring application stability. The evaluation metrics generated by the Simple severity zones and the Severity Value techniques yield a better result (higher utility function values) overall, compared to any of the other commercially inspired techniques. Moreover, other Severity-based techniques also demonstrate at least equal and in general better results (than the commercially inspired techniques) in the isolated testcases which are not covered by the two best techniques. We consider these advantages as a direct contribution to the elasticity capabilities of any cloud application facing challenging workloads.

The superiority of some techniques over others which is underlined above is attributed to their design, and the choice of the utility function. For example, if a technique sacrifices availability to reduce overprovisioning it will have a reduced utility function score if Eq. 11 is used. In turn, the performance of each technique in each of the performance metrics is directly related to its decisions to spawn or deallocate processing instances.

Notwithstanding, the definition of the Severity value can provide an aggregate view of the current situation, for a number of metrics which violate a given elasticity rule. It also decouples the detection of a situation, from the adaptation which will be triggered. This enables the creation of hybrids which can exploit threshold-based rules as a first stage before triggering another adaptation method, e.g., control loops. When the number of n is relatively small (e.g., $n < 10$), the calculation of Severity is very fast. The most important advantage of calculating the Severity value, however, is that it provides a uniform way to measure the importance of a situation, when multiple metrics are involved.

We evaluated several techniques which used no input from the DevOps other than the threshold value and the direction of the scaling. Note that different options could have been considered for some techniques. For example, we could have examined a flavour of the Simple Threshold technique adding or removing 3 instances, or a flavour of the Relative severity zones technique using $k = 0.2$ rather than $k = 0.1$. Hence, the tables which are included in the Evaluation results section serve not as an exhaustive comparison, but rather as an indicator of the suitability of each technique for intense workloads.

Overall, the results appearing in Table 10 are dependent on the datasets which were used and on the design of the experiments. However, we hold that we considered indicative cases by using multiple workloads, spawn delays and threshold pairs. Concerning the evaluation itself and the time intervals which were selected, we underline that although seconds were used for the purposes of comprehensibility, the results would

have been unchanged if the time unit used in the workloads and the formulation of rules was changed to minutes, hours, or by any other proportionate factor. Consequently, even if a workload was less intense but followed the same pattern the same results would still be observed if the windows of the rules and the cooldown period were also changed proportionately.

As regards the exploitation of the techniques which are illustrated in this work in applications from industry (e.g., based on Kubernetes), we propose that they are first implemented in any programming language. Then they should be configured to interface with the programmatic APIs of the platforms or the cloud provider(s) on which the application is deployed to scale it as necessary. In order to produce informed autoscaling decisions, a suitable metric monitoring mechanism will be necessary to feed the values for each metric to the technique at the desired frequency (the tools provided by the cloud provider(s) could also be sufficient).

Conclusions & Further work

In this work we presented a novel algorithmic approach aiding the adaptation of cloud applications - Severity. The DevOps can select any number and any type of monitoring variables – whether bounded or unbounded – to determine the need for an adaptation. The resulting elasticity rules enable finer-grained adaptation decisions compared to rules featuring a single static threshold per rule, or a single scaling action. Besides, elasticity rules are easy to create, easy to process, easy to understand and transfer between applications, and do not require an extensive training period for the system. Based on these elasticity rules, we are able to detect situations, calculate their Severity and propose adaptations which can lead to better application performance. Our approach has been proved to be consistent after its evaluation on four distinct workload types, with a variable delay between deployments. We demonstrated the beneficial results of using the Simple severity zones and Severity value techniques to handle two, three and four monitoring attributes against adaptation techniques representative of those offered by commercial vendors (with the exception of machine-learning based techniques). The implementation of the Situation Detection Mechanism, able to parse rules, converting them to a format understandable by the Siddhi complex processing engine and producing output based on the Simple severity zones technique indicates the feasibility of integrating our approach with existing solutions from the industry.

In the future, the capability to simulate interdependent workloads and the support for more elaborate work allocation schemes (e.g., workload per API request) can lead to finer-grain experiments, and permit a better understanding of the suitability of certain adaptation techniques for particular processing topologies.

Further, the approach which has been outlined above currently only considers horizontal scaling of resources. However, vertical scaling is also very important in the field of cloud application adaptation. This direction of work can noticeably decrease the time required to adapt the application, and therefore needs to be carefully examined.

Another research direction involves adding proactivity to the estimation, the adaptation of the adaptation rules themselves and the evolution of the presented techniques. For example, we have worked on an idea which evolves the concept of Severity-zone based rules, to Severity-zone “Feedback rules”, using previous adaptations to guide successive ones. Alternatively, the thresholds of Severity zones can be changed, or Severity zones can be split into smaller areas, when a critical number of situation samples have been detected in them, followed by a modification of the adaptation action for each area as appropriate.

Moreover, while our approach considers rules provided by the DevOps as the primary input, techniques such as association rules [40, 41] have been implemented, allowing the discovery of new rules, based on the received monitoring data [42]. These statements can subsequently be used to automatically create new rules which would have helped the application to improve its response, had they been introduced in the past.

Besides, the usage of a fixed (and in some cases small) number of zones can lead to only a part of the Severity zones getting activated, due to the nature of a particular workload. In these cases, a greater resolution is more necessary, and approaches such as Q-learning can be more appropriate.

Abbreviations

FaaS: Functions as a Service; HPA: Horizontal Pod Autoscaler; TOSCA: Topology and orchestration specification for cloud applications; VM: Virtual machine; PM: Physical machine; SDM: Situation Detection Mechanism; RARecom: Resources Adaptation Recommender; ASV: Absolute severity value; MACL : Maximum attribute control loop; NAS: Normalized absolute severity; NASCL: Normalized absolute severity control loop; SSZ: Simple severity zones; RSZ: Relative severity zones; SV: Severity value; ST: Simple threshold

Authors’ contributions

AT defined the concept of Severity, executed the State of the Art analysis, implemented the software systems and wrote the illustrative scenario, while also performing the evaluation of the approach and the overall synchronization of the contents of the manuscript. All authors helped in the definition of the concept of Severity. YV helped in the definition of the SSZ and RSZ techniques. NP provided one of the figures used in the illustrative scenario. YV, DA, GM directed the evaluation of the approach and contributed to the improvement of the quality of the paper, providing valuable comments. The authors read and approved the final manuscript.

Funding
Research reported in this paper has received funding from the European Union's Horizon 2020 Research and Innovation program under grant agreements Prestocloud No 732339 and Morphemic No. 871643. The work presented here reflects only the authors' view and the European Commission is not responsible for any use that may be made of the information it contains.

Availability of data and materials
The data which pertains to the experimentation with the techniques which have been defined as part of this work, can be found in our online repository.
Adaptation technique performance using 2, 3 and 4-metric workloads. http://imu.ntua.gr/static/workloads/ . Accessed 06 July 2021.

Declarations

Competing interests
The authors declare no competing interests.

Author details
[1]Information Management Unit (IMU), Institute of Communication and Computer Systems, Athens, Greece. [2]National Technical University of Athens (NTUA), Athens, Greece. [3]Department of Business Administration, Athens University of Economics and Business, Athens, Greece. [4]Department of Informatics, University of Piraeus, Piraeus, Greece.

Received: 3 December 2020 Accepted: 12 July 2021
Published online: 21 August 2021

References

1. Gartner Newsroom (Press Releases). Available online: https://www.gartner.com/en/newsroom/press-releases/2020-11-17-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-18-percent-in-2021. Accessed 7 July 2021
2. Topology and orchestration specification for cloud applications version 1.2, OASIS Standard. Available online: https://docs.oasis-open.org/tosca/TOSCA/v2.0/TOSCA-v2.0.html. Accessed 7 July 2021
3. Gandhi A et al (2014) Adaptive, model-driven autoscaling for cloud applications. In: 11th International Conference on Autonomic Computing. ICAC, p 14
4. Pan SJ, Yang Q (2009) A survey on transfer learning. IEEE Trans Knowl Data Eng 22(10):1345–1359
5. Chen T, Bahsoon R, Yao X (2018) A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. ACM Computing Surveys (CSUR) 51:3 61, 2018
6. Ilyushkin A et al (2017) An experimental performance evaluation of autoscaling policies for complex workflows. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering
7. Podolskiy V, Jindal A, Gerndt M (2018) IaaS Reactive Autoscaling Performance Challenges. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD); July 2018; pp. 954–957
8. Podolskiy V, Mayo M, Koay A, Gerndt M, Patros P (2019) Maintaining SLOs of Cloud-Native Applications Via Self-Adaptive Resource Sharing. In Proceedings of the 2019 IEEE 13th International Conference on Self-Adaptive and Self- Organizing Systems (SASO); June 2019; pp. 72–81
9. Lim HC et al (2009) Automated control in cloud computing: challenges and opportunities. In: Proceedings of the 1st workshop on Automated control for datacenters and clouds. ACM
10. Zhu Q, Agrawal G (2012) Resource provisioning with budget constraints for adaptive applications in cloud environments. IEEE Trans Serv Comput 5(4): 497–511. https://doi.org/10.1109/TSC.2011.61
11. Ashraf A et al (2012) Feedback control algorithms to deploy and scale multiple web applications per virtual machine. In: 2012 38th Euromicro Conference on Software Engineering and Advanced Applications. IEEE
12. Lorido-Botran T, Miguel-Alonso J, Lozano JA (2014) A review of auto-scaling techniques for elastic applications in cloud environments. J Comput 12(4): 559–592. https://doi.org/10.1007/s10723-014-9314-7
13. Arkian H et al (2021) Model-based Stream Processing Auto-scaling in Geo-Distributed Environments. In: ICCCN 2021-30th International Conference on Computer Communications and Networks
14. Taherizadeh S, Stankovski V (2019) Dynamic multi-level auto-scaling rules for containerized applications. Comput J 62(2):174–197. https://doi.org/10.1093/comjnl/bxy043
15. Overview of autoscale with Azure virtual machine scale sets. Available online: https://docs.microsoft.com/enus/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-autoscale-overview. Accessed 7 July 2021
16. Oracle Cloud Infrastructure Documentation – Autoscaling. Available online: https://docs.cloud.oracle.com/enus/iaas/Content/Compute/Tasks/autoscalinginstancepools.htm. Accessed 7 July 2021
17. Taherizadeh S, Grobelnik M (2020) Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications. Adv Eng Softw 140:102734. https://doi.org/10.1016/j.advengsoft.2019.102734
18. Lorido-Botrán, Tania, et al. Comparison of Auto-Scaling Techniques for Cloud Environments. 2013
19. Vaquero LM, Morán D, Galán F, Alcaraz-Calero JM (2012) Towards runtime reconfiguration of application control policies in the cloud. J Netw Syst Manag 20(4):489–512. https://doi.org/10.1007/s10922-012-9251-3
20. Galante G; Bona LCE. Constructing Elastic Scientific Applications Using Elasticity Primitives. In Computational Science and Its Applications – ICCSA 2013; Murgante B, Misra S, Carlini M, Torre CM, Nguyen HQ, Taniar D, Apduhan BO, Gervasi O, Eds.; Lecture Notes in Computer Science. Berlin: Springer. 2013;975:281–294. ISBN 978-3-642-39639-7
21. Copil G et al (2013) Multi-level elasticity control of cloud services. In: International Conference on Service-Oriented Computing. Springer, Berlin
22. Copil G et al (2016) rSYBL: a framework for specifying and controlling cloud services elasticity. ACM Transact Internet Technol 16(3):18
23. Ferretti S et al (2010) Qos–aware clouds. In: 2010 IEEE 3rd International Conference on Cloud Computing. IEEE
24. Trihinas D et al (2017) Improving rule-based elasticity control by adapting the sensitivity of the auto-scaling decision timeframe. In: International Workshop on Algorithmic Aspects of Cloud Computing. Springer, Cham
25. Dutreilh X et al (2010) From data center resource allocation to control theory and back. In: 2010 IEEE 3rd international conference on cloud computing. IEEE
26. Ali-Eldin A, Tordsson J, Elmroth E (2012) An adaptive hybrid elasticity controller for cloud infrastructures. In: 2012 IEEE Network Operations and Management Symposium. IEEE
27. Ali-Eldin A et al (2012) Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In: Proceedings of the 3rd workshop on Scientific Cloud Computing
28. Bauer A et al (2018) Chameleon: a hybrid, proactive auto-scaling mechanism on a level-playing field. IEEE Transact Parallel Distribut Syst 30(4):800–813
29. Ramirez YM, Podolskiy V, Gerndt M (2019) Capacity-driven scaling schedules derivation for coordinated elasticity of containers and virtual machines. In: 2019 IEEE International Conference on Autonomic Computing (ICAC). IEEE
30. Tamiru MA et al (2020) An Experimental Evaluation of the Kubernetes Cluster Autoscaler in the Cloud. In: 2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE
31. Rzadca K et al (2020) Autopilot: workload autoscaling at Google. In: Proceedings of the Fifteenth European Conference on Computer Systems
32. Machine learning predictive scaling for EC2. Available online: https://aws.amazon.com/ru/blogs/aws/new-predictive-scalingfor-ec2-powered-by-machine-learning. Accessed 6 July 2021
33. Alsmeyer G. Chebyshev's Inequality. In International Encyclopedia of Statistical Science; Lovric M, Ed.; Berlin: Springer. 2011. pp. 239–240. ISBN 978-3-642-04897-5
34. Narangoda IL et al (2011) Siddhi: A second look at complex event processing architectures. In: GCE''11 Proceedings of the ACM workshop on Gateway computing environments, vol 10, pp 2110486–2110493
35. Papageorgiou N et al (2019) Situation Detection on the Edge. In: Workshops of the International Conference on Advanced Information Networking and Applications. Springer, Cham
36. Barnawi A, Sakr S, Xiao W, al-Barakati A (2020) The views, measurements and challenges of elasticity in the cloud: a review. Comput Commun 154: 111–117. https://doi.org/10.1016/j.comcom.2020.02.010
37. Simic V, Stojanovic B, Ivanovic M (2019) Optimizing the performance of optimization in the cloud environment–an intelligent auto-scaling approach. Futur Gener Comput Syst 101:909–920. https://doi.org/10.1016/j.future.2019.07.042
38. Adaptation technique performance using 2, 3 and 4-metric workloads. Available online: http://imu.ntua.gr/static/workloads/. Accessed 6 July 2021
39. Tsagkaropoulos A et al (2018) Challenges and Research Directions in Big Data-driven Cloud Adaptivity. CLOSER

40. Kacprzyk J, Zadrożny S (2016) Linguistic summarization of the contents of web server logs via the ordered weighted averaging (OWA) operators. Fuzzy Sets Syst 285:182–198. https://doi.org/10.1016/j.fss.2015.07.020
41. Coulibaly L, Foguem BK, Tangara F (2020) Rule-based machine learning for knowledge discovering in simulated weather data. In: Future Generation Computer Systems
42. Grechanik M et al (2016) Enhancing rules for cloud resource provisioning via learned software performance models. In: Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering

Publisher's Note