

Severity: A QoS-aware approach to cloud application elasticity

Andreas Tsagkaropoulos^{a,b1},

Yiannis Verginadis^{a,c}, Nikos Papageorgiou^{a,b}, Fotis Paraskevopoulos^{a,b}, Dimitris Apostolou^{a,d} and
Gregoris Mentzas^{a,b}

^a*Information Management Unit (IMU), Institute of Communications and Computer Systems, Athens, Greece*

^b*National Technical University of Athens (NTUA), Athens, Greece*

^c*Department of Business Administration, Athens University of Economics and Business, Athens, Greece*

^d*Department of Informatics, University of Piraeus, Piraeus, Greece*

{atsagkaropoulos, jverg, npapag, fotisp, gmentzas}@mail.ntua.gr, dapost@unipi.gr

¹ Corresponding author

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 efficacy 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

1. 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 API's 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.

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 topology. 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.

2. Cloud application elasticity

Decision-making approaches for cloud autoscaling systems are based in general either or rule-based control, or Control Theory and Search-based optimization [5]. The rule-based adaptation approach is one of the simplest and more intuitive approaches which can be followed to scale a cloud topology. 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.

In [6] 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 [7] and Oracle Cloud Infrastructure [8]), the DevOps should either enter a number of rules that scale out/in the current topology by a predefined number of instances (or percentage of the topology). 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 [6], 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 seconds 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 [6] (in terms of the rate of change of the absolute values of the workload), stress-testing the performance of all techniques.

In [9], 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. [10], 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 [6],[9] and [10] are complementary to our approach and can be used to provide an enhanced yet more complex system.

Vaquero et al [17], Galante et al. [18] and Copil et al. [19],[20] have proposed rule-based frameworks, which either rely on user input to calibrate the adaptation actions by manually setting the scaling action as in [17],[18] or always using the same adaptation event (e.g add one VM instance) to keep the desired monitoring attributes to acceptable levels as in [20]. 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. [21], 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 [22], Trihinas et al enhance the basic adaptation support offered by the previous 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. [23] 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.

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.

Control theoretic approaches (e.g [24],[25],[26]) 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 [11] comment on queueing theory models used to horizontally scale a topology, that they suffer from being tightly bound to the workload and the processing topology for which they have been created. As a result, they need to be recalculated when either change.

The complex calculations associated with the optimization of the calculation of Severity Zone boundaries, are based on a simple Markov Chain Monte Carlo simulation. In Section 3.3 we demonstrate that a rule can indicate that an equal partitioning of a multi-dimensional hypercube is necessary, using hyperspherical shells. The calculation of complex integrals which is needed in this – and in more complex cases – is difficult to perform in an analytic fashion. As a result, a Markov Chain Monte Carlo simulation is used which is a well-established technique in the field of engineering.

3. Cloud application elasticity with elasticity rules

3.1.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. 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.

```
For component_id = component_id
if (attribute_1)  $\leq$  attribute_1 value and (attribute_2)
     $\leq$  attribute_2 value and ... and (attribute_n)  $\leq$  attribute_n value
    within Timewindow = Timewindow
and Cooldownperiod has passed from previous adaptation
then Scale_out / Scale_in
```

Listing 1: Elasticity rule format

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 minutes), and the cooldown period which should elapse between two triggerings of the rule.

An instantiated example of a elasticity rule appears in Listing 2:

```
For component_id = VideoTranscoder
if AverageCPUcluster > 70% and AverageRAMcluster > 70%
    within Timewindow = 10 minutes
and 30 minutes have passed from previous adaptations
then Scale_out
```

Listing 2: Elasticity Rule example

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’.

3.2.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 cloud topology, 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 topology should be made (i.e. more VMs should be added/removed).

The Severity of any detected situation $V_{violating} = (v_1, v_2, \dots, v_n)$ is determined as shown in Equation 1.

$$Severity(V_{violating}) = \sqrt{\sum_{i=1}^n w_i \cdot (Normalized(v_i))^2}$$

Equation 1: Calculation of the Severity of a situation

where v_i are the individual, threshold-violating QoS attribute values comprising the particular situation, w_i are their respective weights and n is the number of attributes included in the triggered elasticity rule. For each of the v_i values 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 (Equation 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 v_i - threshold t_i pair in the rule, the threshold, the normalization formula in Equation 2 is used in cases of attributes that need to be greater than their threshold and Equation 3 is used in case of attributes that need to be less than their threshold.

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

Equation 2: Variable normalization in the greater-than case

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

Equation 3: Variable normalization in the less-than case

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 Equation 2 and Equation 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 . We use Chebyshev's equation[16], shown in Equation 4:

$$\Pr(|X - \mu| \geq k \cdot \sigma) = \frac{1}{k^2}$$

Equation 4: Chebyshev's equation

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 Equation 4, and solving for k we determine that $k=5$. 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 $\mu = 200\text{msec}$, with a standard deviation of 30msec . Then, the probability of measuring a response time X , being retarded more than $5 \cdot 30\text{msec} = 150\text{ msec}$ from the expected value (200msec) is less than 4%. We can then estimate the upper bound of the distribution to be $200+150=350\text{msec}$ with 96% certainty.

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.

3.3. Severity Zone Calculation

Although Severity provides an assessment of the Severity of a situation, we need a means 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. For this reason, we introduce ‘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).

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}

Table 1 Severity zones bounds

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 topology. 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, 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.

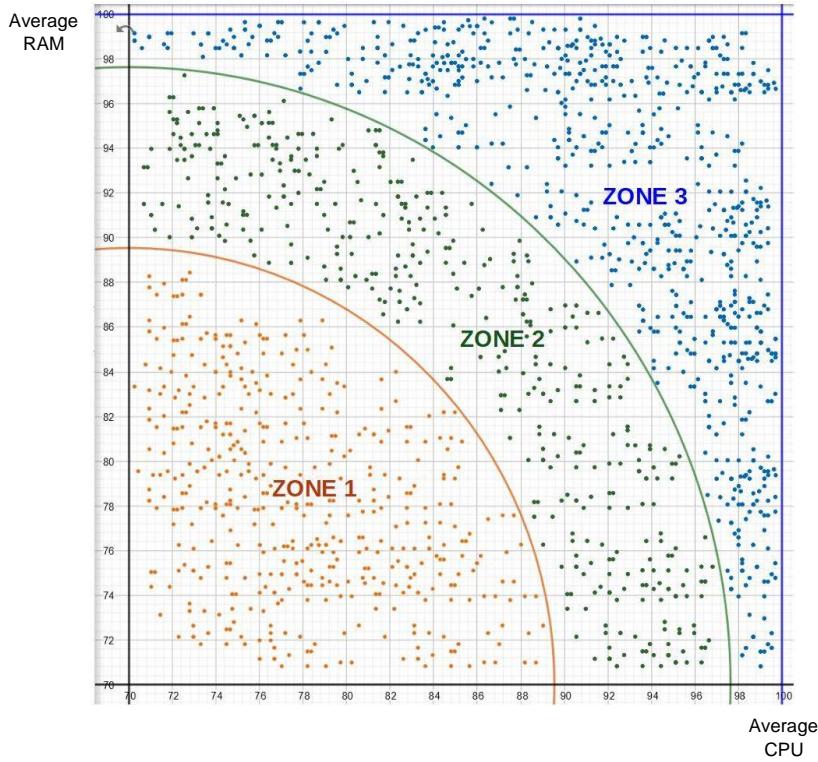


Figure 1 The situation space of the example Elasticity rule of Listing 2, split into 3 equal Severity Zones

Such a solution is possible, if we consider a solution based on a random simulation. For this purpose, we simulate the normalized monitoring attribute values for situations having k normalized attributes, by retrieving random points 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 $s_i \in [0,1] \times [0,1] \times \dots \times [0,1]$. Each point s_i reflects the values of the monitoring attributes of a possible detected situation. The choice of each 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:

$$z = \left\lfloor \frac{p}{m} \right\rfloor$$

Equation 5 Number of random points per Severity zone area (z)

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.

4. 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 of Severity. Sections 4.1 and 4.2 describe two techniques which are based on the 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.

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.

4.1. Simple threshold

The Simple threshold technique is inspired by the offerings of major cloud providers, and the THRES technique described in [6]. 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 appears in Equation 6 and Equation 7.

$$ni = ci + 1$$

Equation 6 New instances determined for a scale out rule using the Simple threshold technique

$$ni = ci - 1$$

Equation 7 New instances determined for a scale in rule using the Simple threshold technique

4.2. 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. 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 Equation 8.

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

Equation 8 New instances determined for a scale out rule using the Maximum attribute control technique

4.3. 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 Equation 2 and Equation 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 shown in Equation 9 and Equation 10.

$$ni = ci(1 + as)$$

Equation 9 New instances determined for a scale out rule using the Absolute severity technique

$$ni = ci(1 - as)$$

Equation 10 New instances determined for a scale out rule using the Absolute severity technique

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. As mentioned above, **as** reflects the absolute Severity value.

As an example, if the absolute Severity of a scale out rule was calculated to be 1.5, and the current number of instances is 10, from Equation 9 the new number of instances will be 25, meaning that 15 new instances will be spawned.

4.4. Normalized absolute severity

The Normalized absolute severity value technique tries to stabilize the topology around the normalized absolute Severity value – which is obtained by dividing the absolute

Severity value with the maximum possible Severity value. The new number of instances after an adaptation is shown in Equation 11 and Equation 12.

$$ni = ci \left(1 + \frac{as}{ms}\right)$$

Equation 11 New instances determined for a scale out rule using the Normalized absolute severity technique

$$ni = ci \left(1 - \frac{as}{ms}\right)$$

Equation 12 New instances determined for a scale in rule using the Normalized absolute severity technique

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 Equation 12 the new number of instances will be 4, meaning that 6 of the instances will be deactivated.

4.5.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. The new number of instances after an adaptation is shown in Equation 13 and Equation 14:

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

Equation 13 New instances determined for a scale out or a scale in rule using the Normalized absolute severity control loop technique

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

Equation 14 New instances determined for a scale in rule using the Normalized absolute severity control loop technique

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 upper threshold Severity for this rule is 0.6 and the current number of instances is 10, from Equation 14 the new number of instances will be 5, meaning that 5 instances will be deactivated.

4.6.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 current topology. The flavor 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 topology as appropriate. The new number of instances after an adaptation is shown in Equation 15 and Equation 16:

$$ni = ci + sz$$

Equation 15 New instances determined for a scale out rule using the Simple severity zones technique

$$ni = ci - sz$$

Equation 16 New instances determined for a scale in rule using the Simple severity zones technique

As an example, if a topology has 4 instances and the Severity zone of a situation triggered by a scale out rule was found to be 2, from Equation 15 the new number of instances will be $4+2=6$.

4.7.Relative severity zones

The Relative severity zones technique uses the concept of Severity zones to add (or remove) a percentage of the topology to the current number of instances. 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 the topology will deactivate all available instances (if no other specific handling of this issue occurs). The flavor 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 Equation 17 and Equation 18:

$$ni = ci (1 + k \cdot sz)$$

Equation 17 New instances determined for a scale out rule using the Relative severity zones technique

$$ni = ci (1 - k \cdot sz)$$

Equation 18 New instances determined for a scale in rule using the Relative severity zones technique

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.

4.8. 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 Equation 2 and Equation 3. The updated number of instances based on this technique is shown in Equation 19 and Equation 20:

$$ni = ci(1 + s)$$

Equation 19 New instances determined for a scale out rule using the Severity value technique

$$ni = ci(1 - s)$$

Equation 20 New instances determined for a scale in rule using the Severity value technique

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.

4.9. 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 result. The new number of instances after the application of this technique appears in Equation 21 and Equation 22, for scale out and scale in rules respectively.

$$ni = ci\left(1 + \frac{s}{ms}\right)$$

Equation 21 New instances determined for a scale out rule using the Severity value technique

$$ni = ci\left(1 - \frac{s}{ms}\right)$$

Equation 22 New instances determined for a scale in rule using the Severity value technique

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.

5. Illustrative scenario implementation

In this section, we will provide a walkthrough of the situation detection process, from the monitoring data published by the topology to the adaptation which is decided. We will focus in 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:

```
For fragid = VideoTranscoder  
if AverageCPUcluster > 70% and AverageRAMcluster > 70%  
    within Timewindow = 10 minutes  
    and 30 minutes have passed from previous adaptations  
        then Scale_out
```

Listing 3 Elasticity Rule example

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 minutes, a new scale out adaptation action decision should be issued – provided that no previous adaptation event has occurred in the last 30 minutes (cooldown period).

5.1.Situation Detection

We consider that at a certain time point, a new observation is detected, indicating that over the last 10 minutes, the average values for CPU and RAM were 92% and 71% respectively. Moreover, no adaptation event has occurred in the last 30 minutes. 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:

$$\begin{aligned} \text{Normalized}(CPU_1) &= \frac{\text{abs}(92 - 70)}{\text{abs}(100 - 70)} = \frac{22}{30} \sim 73.3\% \\ \text{Normalized}(RAM_1) &= \frac{\text{abs}(71 - 70)}{\text{abs}(100 - 70)} = \frac{1}{30} \sim 3.3\% \end{aligned}$$

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:

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

Listing 4 The calculated Severity of the situation

5.2.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 ($\text{AverageCPU}_{\text{cluster}}$ and $\text{AverageRAM}_{\text{cluster}}$). Following the random point generation and sorting, the Severity zones are determined to have the upper bounds indicated in Table 2.

Severity zone	Calculated upper bound
1	0.651
2	0.921
3	1.414

Table 2: Upper bounds for three Severity zones with two attributes

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 CPU consumption is indicated in the vertical axis, while the reported average RAM consumption is indicated in the horizontal axis. Following the upper bound calculation method presented above, the three Severity zones are depicted in Figure 2.

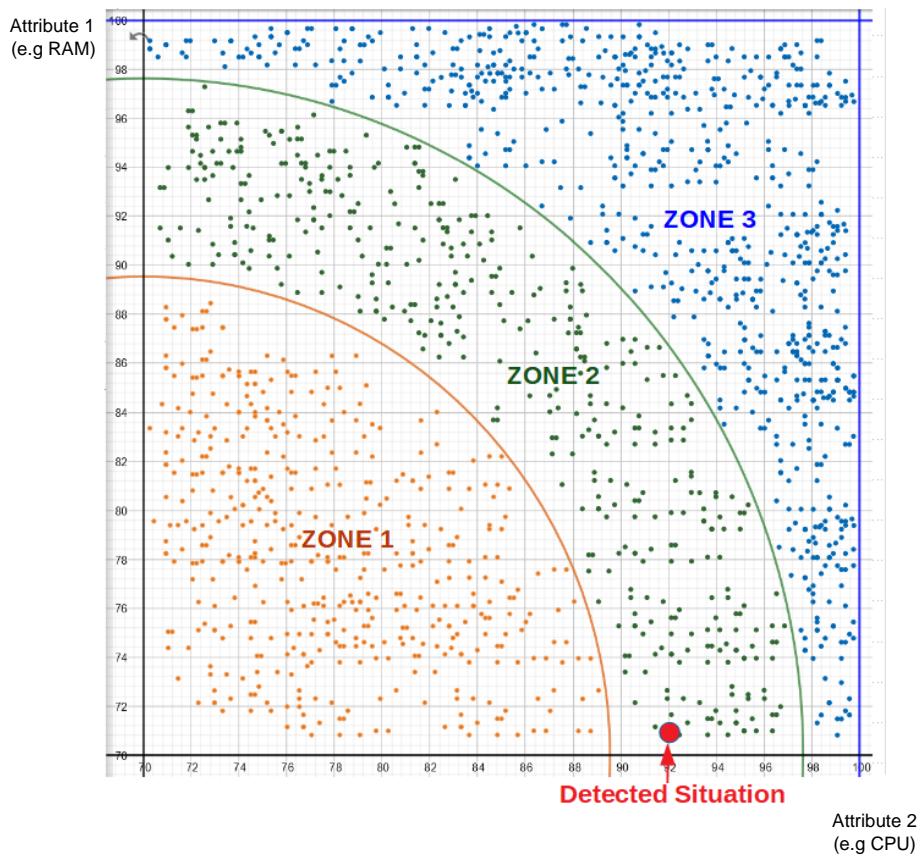


Figure 2: Severity zones in the case of two metrics and three zones

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 Figure 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.

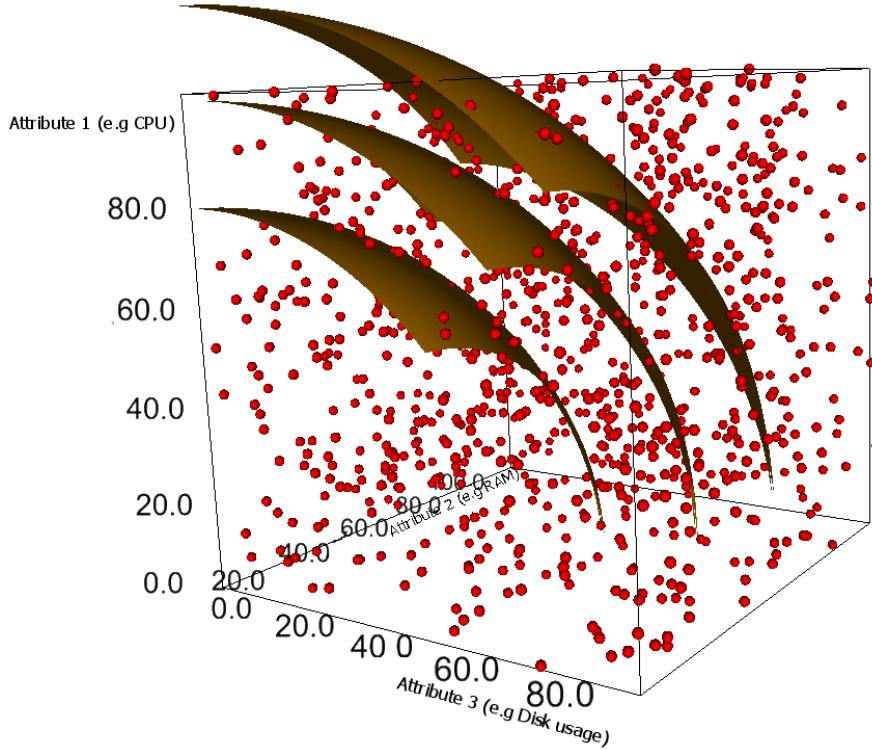


Figure 3: Severity zones in the case of three metrics and four zones

6. 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 Figure 4:

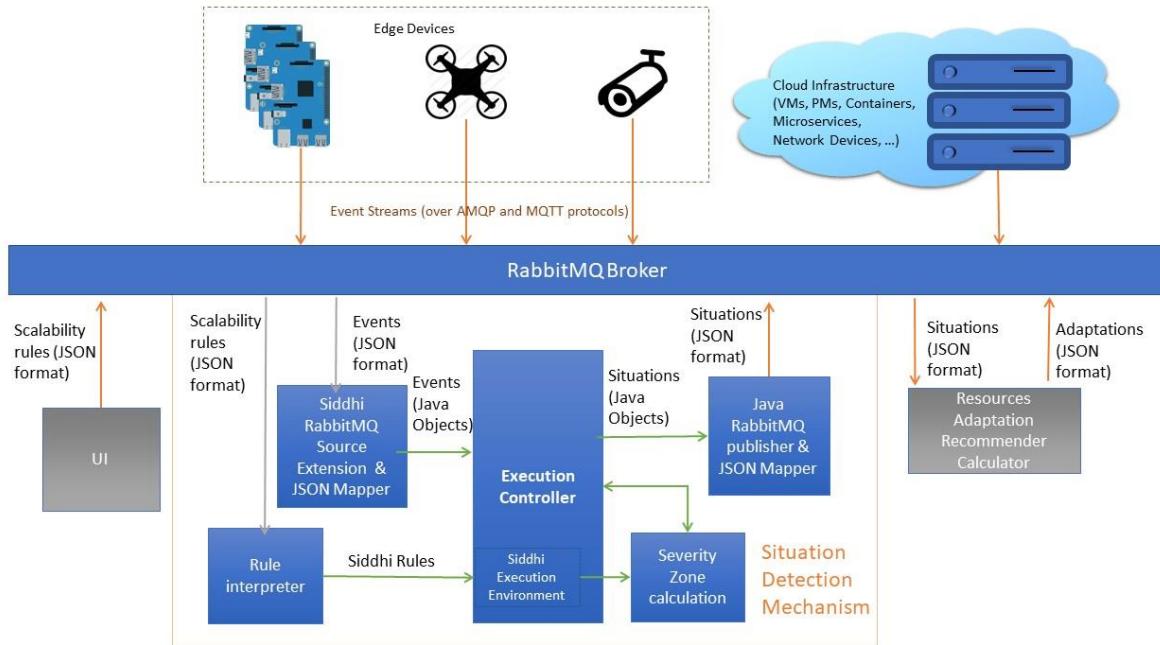


Figure 4 Architecture of prototype situation detection implementation

The architecture of the Situation Detection Mechanism is structured around the usage of a common message bus, in this case the RabbitMQ² 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 SDM can handle the monitoring of a processing topology 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 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. Once a elasticity rule is received, the SDM subscribes to the RabbitMQ broker which is directly connected with the infrastructure forming the processing topology. Then, it can start consuming monitoring events which are related to the metrics of the rule.

The *Rule Interpreter* module undertakes the conversion of elasticity rules to queries understandable by the Siddhi streaming input processor [27]. 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).

² <https://www.rabbitmq.com/>

```

define stream scalability_rule_stream (avg_cpu_perc double, avg_mem_perc double, fragid string,  

res_inst string);

from serverStream#window.timeBatch(600 sec) [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;

```

Listing 5: Siddhi query created for example elasticity rule

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.

The choice to use Siddhi builds upon our previous work [28]. 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. 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).

7. Evaluation

7.1.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 [29].

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 [6]. 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 the common approaches which are found in commercial rule-based 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 3:

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

Table 3 The upper and lower thresholds of the elasticity rules which were used

The maximum number of attribute values which were examined simultaneously was 4, although the 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 3 to handle the workloads presented in Figures 5-8, reconfiguring the topology and collecting metrics on the performance of the application. The time window of the rules was set equal to one-fifth of the VM spawn duration with a minimum duration of 3 seconds (i.e. 3, 3, 6 and 12 seconds for the 0, 15, 30 and 60 second spawn intervals which were tested). A cooldown period of 10 seconds 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 amount 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 4.

Benchmarking metric	Description
Availability	The percentage of time for which the workload pattern for a particular metric was not more the processing capacity of the topology. 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 topology 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 topology 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.

Table 4 Description of the benchmarking metrics used for the evaluation of Severity-based techniques

Other approaches have also been using similar metrics in their experiments. For example [6] and [10] examined the number of containers and VMs respectively (which can be related to cost) and the response time for different techniques, while [30] examined cost and execution time (which can be converted to a question between cost and availability). In a thorough review of cloud elasticity [29], 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 all four 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). 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. [6]) 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.

4-metric workloads

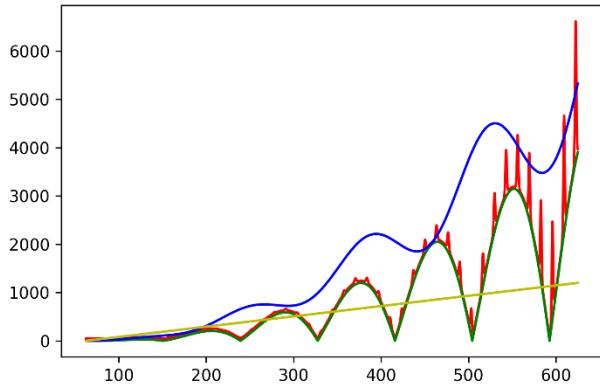


Figure 5 Periodically increasing workload, with spikes

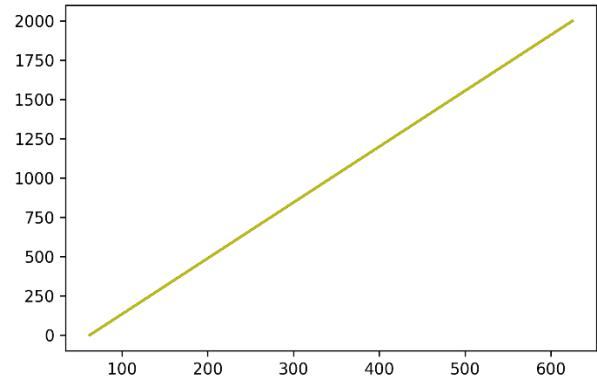


Figure 6 Linearly increasing workload. Due to overlapping of the values, only the yellow line is visible

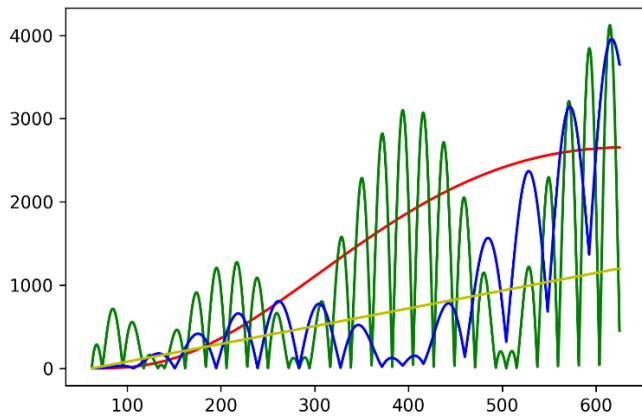


Figure 7 Periodically increasing workload, with fluctuations

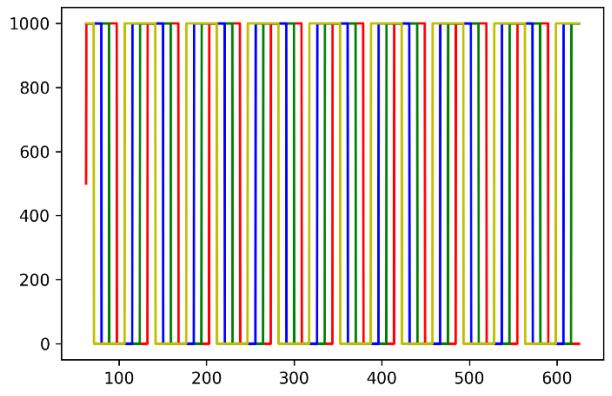


Figure 8 Polarized workload

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 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.

7.2. 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 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 topology, 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 5, 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 techniques. In Figure 9 and Figure 10 we included data collected for 2 and 3-metric workloads respectively, using a particular VM spawn delay, regarding the Simple severity zones and Severity value techniques and the Maximum attribute control loop techniques. The Simple severity zones and Severity value techniques 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 3. 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 [31]. All values in Table 5 reflect percentages, except for the number of topology 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 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.

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 -9.1	-50.5 25.4	-43.4 31.8	-61.2 9.3	19.0 47.1	25.0 64.4	23.2 52.6	20.0 41.2	3.9 50.3	17.9 87.3	33.3 75.5	24.2 68.2	16 63	16 62	27 46	15 38
	Linearly increasing	10.1 49.9	-4.6 42.5	22.5 128.1	19.8 92.8	61.2 100.0	40.0 98.9	47.2 95.9	41.6 86.7	0.2 82.1	13.4 90.5	13.4 92.3	15.3 93.1	23 118	11 65	11 52	11 46
	Periodically increasing with fluctuations	-9.4 28.6	-74.4 45.0	-36.4 57.7	-30.5 14.7	30.3 73.0	12.5 72.3	25.3 66.1	28.4 57.4	0.2 22.7	5.6 57.5	10.4 65.7	15.4 71.2	8 43	8 43	9 42	12 31
	Polarized	-61.9 -57.8	-75.8 -74.0	-79.2 -72.9	-48.2 -23.5	30.6 32.9	12.5 12.5	12.5 46.4	28.1 36.3	29.4 57.2	54.1 74.3	63.5 52.9	33.1 33	32 34	32 40	38 39	
Maximum attribute control loop	Periodically increasing with spikes	-73.6 -34.9	-75.4 -42.9	-72.0 -36.3	-65.3 -18.3	9.2 33.0	9.2 20.4	8.3 24.4	9.9 24.6	33.5 65.0	39.9 75.5	34.7 74.2	36.8 70.0	31 47	30 48	27 46	25 43
	Linearly increasing	10.1 50.5	8.1 48.3	8.8 43.5	-16.3 66.0	100.0 100.0	96.6 99.1	61.7 95.9	46.0 87.7	0.2 12.1	22.2 40.0	27.7 89.8	30.1 91.5	23 31	23 31	21 52	21 45
	Periodically increasing with fluctuations	-11.6 31.0	-13.2 26.3	-35.7 26.2	-38.7 40.6	34.7 73.1	32.7 66.5	14.8 63.2	23.1 60.3	3.1 10.5	12.3 24.1	15.3 40.8	24.6 55.0	12 25	13 23	12 32	20 30
	Polarized	-79.7 -70.8	-79.7 -76.5	-79.2 -77.4	-82.2 -44.8	12.5 12.5	12.5 12.5	12.5 29.4	34.9 36.7	54.1 57.5	63.5 74.3	30.1 51.3	32 33	32 34	38 40	31 51	

Table 5 Techniques comparison matrix (4 metric workloads)

For brevity, in Figure 9 and Figure 10 we present using candlestick charts the performance of the Maximum attribute control loop algorithm against the Simple severity zones and Severity value algorithms respectively. Similar to Table 5, we provide the minimum and maximum values for each of the benchmarking metrics – the top edge of each candlestick 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 topology adaptations metric which is an integer. The abbreviations which are used are described in Table 6.

Benchmarking metric	Description
macl_op, ssz_op, sv_op	Overprovisioning of the Maximum attribute control loop, Simple severity zones and Severity value techniques respectively
macl_av, ssz_av, sv_av	Availability of the Maximum attribute control loop, Simple severity zones and Severity value techniques respectively
macl_rg, ssz_rg, sv_rg	Rigidness of the Maximum attribute control loop, Simple severity zones and Severity value techniques respectively
macl_ad, ssz_ad, sv_ad	Number of topology adaptations of the Maximum attribute control loop, Simple severity zones and Severity value techniques respectively

Table 6 Benchmarking metric abbreviations

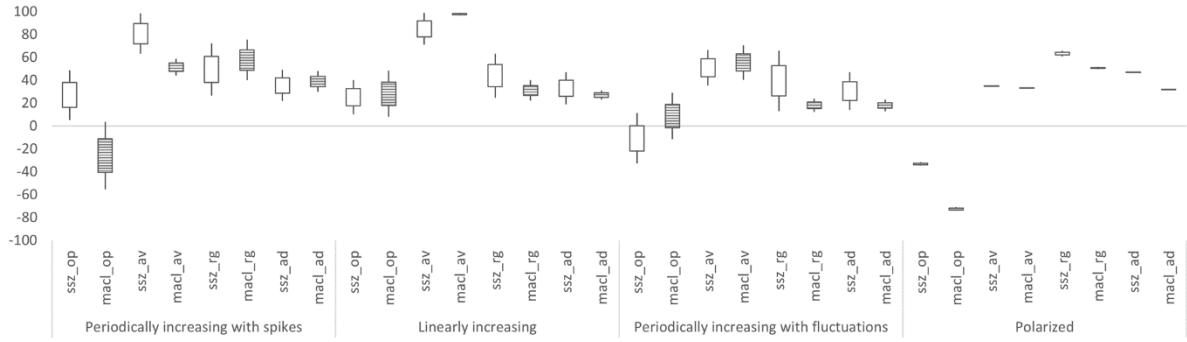


Figure 9 2-metric, 15-delay workload performance of the Simple severity zones and Maximum attribute control loop techniques

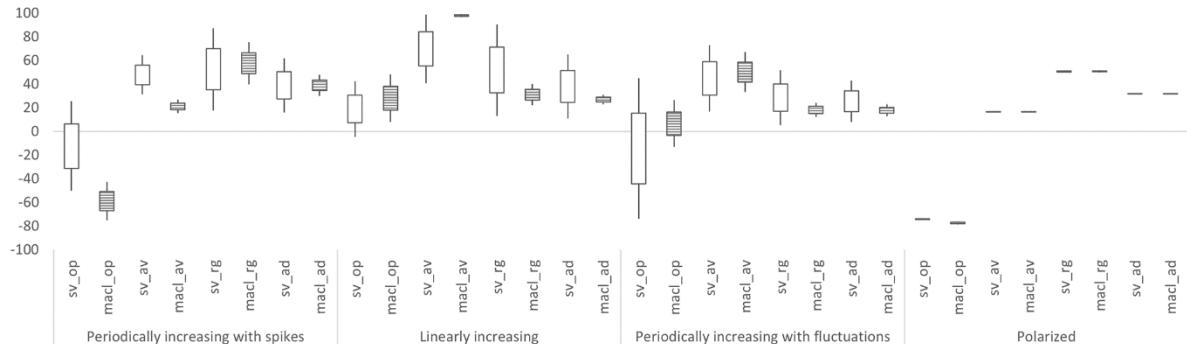


Figure 10 3-metric, 15-delay workload performance of the Severity value and Maximum attribute control loop techniques

Table 5, Figure 9 and Figure 10 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 topology 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 topology adaptations. The utility function using the benchmarking metrics, as well as their weights appear in bold in Equation 23.

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

Equation 23 Example Utility Function

Table 7, Table 8 and Table 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 seconds, respectively. The following abbreviations are used: Absolute severity value (ASV); Maximum attribute

control loop (MACL); 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.

Workload	ASV	MACL	NAS	NASCL	NSV	SV	RSZ	SSZ	ST
Linearly increasing	0.78	<u>0.82</u>	0.78	<u>0.82</u>	0.80	<u>0.82</u>	<u>0.82</u>	<u>0.82</u>	<u>0.82</u>
	0.66	<u>0.77</u>	0.74	<u>0.77</u>	0.76	<u>0.78</u>	<u>0.77</u>	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	<u>0.72</u>	<u>0.72</u>	<u>0.72</u>	<u>0.72</u>	0.63
	0.59	0.59	0.56	0.68	0.68	0.61	0.70	0.67	0.68
	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	<u>0.69</u>	<u>0.69</u>	0.67	0.65	0.52
	0.54	0.65	<u>0.67</u>	0.63	0.60	<u>0.67</u>	0.63	0.66	0.56
	0.63	0.59	0.64	0.57	0.62	<u>0.65</u>	0.57	<u>0.65</u>	0.53
Polarized	0.65	0.60	0.58	0.60	0.56	0.65	0.54	0.57	0.54
	<u>0.57</u>	<u>0.57</u>	<u>0.57</u>	<u>0.57</u>	<u>0.57</u>	<u>0.57</u>	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 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	<u>0.82</u>	0.78	<u>0.82</u>	0.80	<u>0.82</u>	<u>0.82</u>	<u>0.82</u>	<u>0.82</u>
	0.66	<u>0.77</u>	0.74	<u>0.77</u>	0.76	<u>0.78</u>	<u>0.77</u>	<u>0.78</u>	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	<u>0.49</u>
	0.55	0.47	0.55	0.68	0.68	0.61	0.70	0.67	0.68
	0.50	0.48	0.51	0.47	<u>0.59</u>	0.57	0.50	<u>0.59</u>	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	0.49
	0.53	0.63	<u>0.65</u>	0.62	0.59	<u>0.65</u>	0.62	0.64	0.53
	0.62	0.58	0.62	0.56	0.61	<u>0.63</u>	0.56	<u>0.63</u>	0.51
Polarized	<u>0.59</u>	0.53	0.52	0.53	0.49	<u>0.59</u>	0.48	0.51	0.48
	<u>0.50</u>	<u>0.51</u>	0.50	<u>0.51</u>	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

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

Workload name	ASV	MACL	NAS	NASCL	NSV	SV	RSZ	SSZ	ST
Linearly increasing	0.78	<u>0.82</u>	0.78	<u>0.82</u>	0.80	<u>0.82</u>	<u>0.82</u>	<u>0.82</u>	<u>0.82</u>
	0.66	<u>0.77</u>	0.74	<u>0.77</u>	0.76	<u>0.78</u>	0.77	<u>0.78</u>	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	0.68	0.68	0.59	0.70	0.67	0.68
	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	<u>0.67</u>	<u>0.67</u>	0.66	0.63	0.49
	0.53	0.63	<u>0.65</u>	0.62	0.59	<u>0.65</u>	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	<u>0.57</u>	0.51	0.50	0.51	0.47	<u>0.57</u>	0.45	0.49	0.45
	<u>0.48</u>	<u>0.48</u>	<u>0.48</u>	<u>0.48</u>	<u>0.48</u>	<u>0.48</u>	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 9 The performance of each technique using 4-dimensional workloads, and a variable spawn delay

From the data which is presented in Table 7, Table 8 and Table 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. Moreover, it is apparent that some techniques are more suitable for particular cloud application settings. 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 seconds. On the contrary, the Simple severity zones technique consistently attains the best results when the spawn delay is 30 and 60 seconds. Moreover, the Severity value technique attains excellent results when the delay is zero 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.

	2-metric workloads	3-metric workloads	4-metric workloads
Maximum improvement over best commercial technique evaluated	3.79%	6.58%	7.22%
Best single-technique improvement over best single commercial technique evaluated	1.54%	5.88%	6.53%
Best Severity-based technique	Simple severity zones	Severity value	Severity value

Table 10: Improvement of utility function values from 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 thrice able to equal this exact maximum value – which besides was attained by the Normalized absolute severity control loop technique. In Table 7, Table 8 and Table 9 above this appears to happen more often due to rounding.

8. 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 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 [32]. 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.

We demonstrated that using a utility function which prioritizes the correct provisioning and service availability while also favoring application stability, the Simple severity zones and the Severity Value techniques yield a better overall result than any of the other techniques. Moreover, other Severity-based techniques also demonstrate better results 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.

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 flavor of the Simple Threshold technique adding or removing 3 instances, or a flavor of the Relative severity zones technique using $k=0.2$ rather than $k=0.1$. Hence, the tables which are included in the relevant section serve not as an exhaustive comparison, but rather as an indicator of the suitability of each technique for intense workloads.

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 were also changed proportionately.

9. 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. 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 [33],[34] have been implemented, allowing the discovery of new rules, based on the received monitoring data [35]. 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.

10. References

- [1] Gartner Newsroom (Press Releases). <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 01 December 2020.
- [2] Topology and orchestration specification for cloud applications version 1.2, OASIS Standard. <http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/TOSCA-Simple-Profile-YAML-v1.2.html>. Accessed 01 December 2020.
- [3] Gandhi, Anshul, et al. "Adaptive, model-driven autoscaling for cloud applications." 11th International Conference on Autonomic Computing ({ICAC} 14). 2014.

- [4] Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." *IEEE Transactions on knowledge and data engineering* 22.10 (2009): 1345-1359.
- [5] Chen, Tao, Rami Bahsoon, and Xin Yao. "A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems." *ACM Computing Surveys (CSUR)* 51.3 (2018): 61., 2018
- [6] Taherizadeh, Salman, and Vlado Stankovski. "Dynamic multi-level auto-scaling rules for containerized applications." *The Computer Journal* 62.2 (2019): 174-197.
- [7] Overview of autoscale with Azure virtual machine scale sets.
<https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-autoscale-overview> . Accessed 01 December 2020.
- [8] Oracle Cloud Infrastructure Documentation – Autoscaling.
<https://docs.cloud.oracle.com/en-us/iaas/Content/Compute/Tasks/autoscalinginstancepools.htm> . Accessed 01 December 2020.
- [9] Taherizadeh, Salman, and Marko Grobelnik. "Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications." *Advances in Engineering Software* 140 (2020): 102734.
- [10] Lorido-Botrán, Tania, et al. Comparison of Auto-Scaling Techniques for Cloud Environments. 2013.
- [11] Lorido-Botran, Tania, Jose Miguel-Alonso, and Jose A. Lozano. "A review of auto-scaling techniques for elastic applications in cloud environments." *Journal of grid computing* 12.4 (2014): 559-592.
- [12] PrEstoCloud Deliverable D2.3, "Conceptual Architecture". <http://prestocloud-project.eu/index.php/deliverables/>. Accessed 01 December 2020.
- [13] PrEstoCloud Deliverable D6.1, "Architecture of the PrEstoCloud platform". <http://prestocloud-project.eu/index.php/deliverables/>. Accessed 01 December 2020.
- [14] PrEstoCloud Deliverable D5.5, "Resources Adaptation & Data-intensive Application Recommenders – Iteration 1". <http://prestocloud-project.eu/index.php/deliverables/>. Accessed 01 December 2020.
- [15] PrEstoCloud Deliverable D5.1, "Situation Detection Mechanism – Iteration 1". <http://prestocloud-project.eu/index.php/deliverables/>. Accessed 01 December 2020.
- [16] Alsmeyer, Gerold. "Chebyshev's inequality." *International Encyclopedia of Statistical Science* (2011): 239-240.
- [17] Vaquero, Luis M., et al. "Towards runtime reconfiguration of application control policies in the cloud." *Journal of Network and Systems Management* 20.4 (2012): 489-512.
- [18] Galante, Guilherme, and Luis Carlos Erpen Bona. "Constructing elastic scientific applications using elasticity primitives." *International Conference on Computational Science and Its Applications*. Springer, Berlin, Heidelberg, 2013.
- [19] Copil, Georgiana, et al. "Multi-level elasticity control of cloud services." *International Conference on Service-Oriented Computing*. Springer, Berlin, Heidelberg, 2013.

- [20] Copil, Georgiana, et al. "rSYBL: a framework for specifying and controlling cloud services elasticity." *ACM Transactions on Internet Technology (TOIT)* 16.3 (2016): 18.
- [21] Ferretti, Stefano, et al. "Qos-aware clouds." *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE, 2010.
- [22] Trihinas, Demetris, et al. "Improving Rule-Based Elasticity Control by Adapting the Sensitivity of the Auto-Scaling Decision Timeframe." *International Workshop on Algorithmic Aspects of Cloud Computing*. Springer, Cham, 2017.
- [23] Dutreilh, Xavier, et al. "From data center resource allocation to control theory and back." *2010 IEEE 3rd international conference on cloud computing*. IEEE, 2010.
- [24] Lim, Harold C., et al. "Automated control in cloud computing: challenges and opportunities." *Proceedings of the 1st workshop on Automated control for datacenters and clouds*. ACM, 2009.
- [25] Zhu, Qian, and Gagan Agrawal. "Resource provisioning with budget constraints for adaptive applications in cloud environments." *IEEE Transactions on Services Computing* 5.4 (2012): 497-511.
- [26] Ashraf, Adnan, et al. "Feedback control algorithms to deploy and scale multiple web applications per virtual machine." *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2012.
- [27] Narangoda, Isuru Loku, et al. "Siddhi: A second look at complex event processing architectures." *GCE'11 Proceedings of the ACM workshop on Gateway computing environments*. Vol. 10. No. 2110486.2110493. 2011.
- [28] Papageorgiou, Nikos, et al. "Situation Detection on the Edge." *Workshops of the International Conference on Advanced Information Networking and Applications*. Springer, Cham, 2019.
- [29] Barnawi, Ahmed, et al. "The views, measurements and challenges of elasticity in the cloud: A review." *Computer Communications* 154 (2020): 111-117.
- [30] Simic, Visnja, Boban Stojanovic, and Milos Ivanovic. "Optimizing the performance of optimization in the cloud environment—An intelligent auto-scaling approach." *Future Generation Computer Systems* 101 (2019): 909-920.
- [31] Adaptation technique performance using 2, 3 and 4-metric workloads.
<http://imu.ntua.gr/static/workloads/> . Accessed 02 December 2020.
- [32] Tsagkaropoulos, Andreas, et al. "Challenges and Research Directions in Big Data-driven Cloud Adaptivity." *CLOSER*. 2018
- [33] Kacprzyk, Janusz, and Sławomir Zadrożny. "Linguistic summarization of the contents of Web server logs via the Ordered Weighted Averaging (OWA) operators." *Fuzzy Sets and Systems* 285 (2016): 182-198.
- [34] Coulibaly, Lassana, Bernard Kamsu Foguem, and Fana Tangara. "Rule-based machine learning for knowledge discovering in simulated weather data." *Future Generation Computer Systems* (2020).
- [35] Grechanik, Mark, et al. "Enhancing rules for cloud resource provisioning via learned software performance models." *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*. 2016.

Declarations

Availability of data and material

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 02 December 2020.

Competing interests

The authors declare no competing interests.

Acknowledgements

List of 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.

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.