A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems

TAO CHEN, Department of Computing and Technology, Nottingham Trent University, UK, and CERCIA, School of Computer Science, University of Birmingham, UK
RAMI BAHSOON, CERCIA, School of Computer Science, University of Birmingham, UK
XIN YAO, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China, and CERCIA, School of Computer Science, University of Birmingham, UK

Autoscaling system can reconfigure cloud-based services and applications, through various configurations of cloud software and provisions of hardware resources, to adapt to the changing environment at runtime. Such a behavior offers the foundation for achieving elasticity in a modern cloud computing paradigm. Given the dynamic and uncertain nature of the shared cloud infrastructure, the cloud autoscaling system has been engineered as one of the most complex, sophisticated, and intelligent artifacts created by humans, aiming to achieve self-aware, self-adaptive, and dependable runtime scaling. Yet the existing Self-aware and Self-adaptive Cloud Autoscaling System (SSCAS) is not at a state where it can be reliably exploited in the cloud. In this article, we survey the state-of-the-art research studies on SSCAS and provide a comprehensive taxonomy for this field. We present detailed analysis of the results and provide insights on open challenges, as well as the promising directions that are worth investigated in the future work of this area of research. Our survey and taxonomy contribute to the fundamentals of engineering more intelligent autoscaling systems in the cloud.

CCS Concepts: • **Software and its engineering → Cloud computing**; *Software performance*;

Additional Key Words and Phrases: Cloud computing, auto-scaling, resources provisioning, distributed systems, self-aware systems, self-adaptive systems

ACM Reference format:
Tao Chen, Rami Bahsoon, and Xin Yao. 2018. A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. *ACM Comput. Surv.* 51, 3, Article 61 (June 2018), 40 pages.
https://doi.org/10.1145/3190507

This work was supported by the Ministry of Science and Technology of China (Grant No. 2017YFC0804003), Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS201703031748284), and EPSRC (Grant No. EP/J017515/01 and EP/K001523).
Authors' addresses: T. Chen (co-corresponding author), Nottingham Trent University, Nottingham, UK, NG11 8NS and University of Birmingham, Birmingham, UK, B15 2TT; email: t.chen@cs.bham.ac.uk; R. Bahsoon (co-corresponding author), University of Birmingham, Birmingham, UK, B15 2TT; email: r.bahsoon@cs.bham.ac.uk; X. Yao (co-corresponding author), Southern University of Science and Technology, Shenzhen, China, 518055 and University of Birmingham, Birmingham, UK, B15 2TT; email: xiny@sustc.edu.cn.

ACM 0360-0300/2018/06-ART61 $15.00
https://doi.org/10.1145/3190507

1 INTRODUCTION

Modern IT companies, from small business to large enterprises, increasingly leverage cloud computing to improve their profits and reduce costs. Throughout all the Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) levels, one of the pronounced benefits of the cloud is *elasticity*, which reflects the extent to which a system can adapt to the workload fluctuations by adjusting configurations and resource provisioning close to the demand. In certain predictable scenarios where the environmental condition has strong and stable seasonality, the configurations and resources can be approximately specified by human experts in advance. Nevertheless, for many other cases, for example, unexpected workload changes, elasticity can be only enabled by runtime automatic scaling or, simply, autoscaling: *a dynamic process, often operating on a Physical Machine (PM), that adapts software configurations (e.g., threads, connections and cache, etc.) and hardware resources provisioning (e.g., CPU, memory, etc.) on demand, according to the time-varying environmental conditions.* The ultimate goal of autoscaling is to continually optimize the non-functional Quality of Service (QoS) (e.g., response time and throughput) and cost objectives for all cloud-based services[1]; thus their Service Level Agreement (SLA) and budget requirements can be better complied with. In particular, autoscaling systems help to realize elasticity by providing timely and elastic adaptation in scales, which is one of the key benefits of cloud computing that attracts a wide range of practitioners [109, 122]. Comparing with other cloud resources management sytems in general, the autoscaling system has been specifically designed for (i) scaling cloud-based applications in response to dynamic changes in load, uncertainties in operations, handling multitenancy while ensuring Service Level Agreement compliance and so on (in contrast, other resource management tasks, e.g., resource scheduling, often work on planned and deterministic sequence of resource demand); (ii) adapting both the software configurations and hardware resources (and their interplays) that span over all SaaS, PaaS, and IaaS levels, whereas most of the other resource management considers hardware resources and IaaS only; and (iii) taking the QoS for cloud-based applications/services at the centre of the concerned objectives (explicitly or implicitly) while the other resource management tasks often focus on resource utilization.

From the literal meaning of the word "autoscaling," it is obvious that the process is dynamic and requires the system to adapt itself subject to the dynamic and uncertain state of the services being managed and the environment. In such a way, cloud-based services, running on a Virtual Machine (VM) or containers, can be "expanded" and "shrink" according to the environmental conditions at runtime. This characteristic has made autoscaling systems well suited to the broad category of self-aware and self-adaptive systems [47, 56, 128]. However, given the unique characteristics of the cloud, engineering a Self-aware and Self-adaptive Cloud Autoscaling System (SSCAS) poses many challenges, including efficient autoscaling architectural styles, an accurate model to predict the effects of autoscaling decisions[2] on the quality attributes, appropriate granularity of runtime control, and effective tradeoff decision making. In particular, a SSCAS should be able to handle various dimensions of QoS attributes, software configuration and hardware resources, in the presence of ***QoS interference*** [100, 127, 141, 111], where the quality of a single cloud-based service can be influenced by the dynamic behaviors of its neighbors on a PM, i.e., the other co-located services and co-hosted VMs or containers, under the sharing infrastructure of cloud.

In this article, we provide a survey and taxonomy for the landscape of SSCAS research to better understand the state of the arts and to identify the open challenges in the field. In particular, we focus on cloud autoscaling research with respect to the well-known principles of self-awareness [34] and self-adaptively [128] in computing systems, as well as the fundamental

[1]Service could refer to an entire application, or any conceptual part within an application.

[2]Each autoscaling decision is a specific combination of configurations and/or resource provisions that achieves certain outcomes on the targeted objectives.

approaches and techniques that realize them. Broadly speaking, we aim to answer the following research questions: (i) What are the levels of self-awareness and self-adaptivity that have been captured in SSCAS? (ii) What are the architectural patterns used for engineering SSCAS? (iii) What are the approaches used to model the quality related to SSCAS? (iv) What is the granularity of control in SSCAS? (v) What are the approaches used for decision making in SSCAS? In a nutshell, our key findings are as follows:

- *Stimulus-*, *time-*, and *goal-awareness* are the most commonly considered self-awareness levels in current SSCAS research while *self-configuring* and *self-optimizing* are more attractive than the others on self-adaptivity (please refer to Sections 2.3 and 2.2 for their definitions, respectively).
- It was found that the general and simple feedback loop architectural pattern (and its variations) has been prominent for engineering SSCAS.
- Analytical and machine-learning-based modeling have been the most widely used approaches for modeling the effects of autoscaling decision on quality attributes. But, surprisingly, systematic selection of model's input features and QoS interference are rarely considered for SSCAS.
- The level of service/application is the most popular granularity of control for SSCAS.
- Explicit optimization driven decision making is the most commonly used approach in SSCAS, but most of them have assumed single objective or using weighted sum aggregation of objectives. Further, we noted that QoS interference is again absent in many studies.

In addition, we obtain the following insights on the open challenges for future research of the field:

- There is a lack of considering required knowledge and its representation for SSCAS architecture with respect to the principles of self-awareness [34], thus urging further investigations. This can help to reason about and prevent improper design decisions, leading to better self-adaptivity.
- Despite that QoS interference has been found to be an important issue [100, 127, 141, 111], it is often overlooked in both the QoS modeling and decision-making aspects of SSCAS. Therefore, we call for novel and effective approaches to manage and mitigate QoS interference in the cloud.
- Most studies attempt to scale hardware resources at the IaaS level only. However, a mature SSCAS should additionally consider the software configurations related to the cloud-based services and their interplay with the hardware resources, as found in recent studies [39, 152, 111].
- Instead of using fixed granularity of control (i.e., the boundary of decision making is on each service/application, VM/container, PM or cloud), future SSCAS should consider more flexible ones, e.g., dynamic and hybrid levels, as discovered in recent studies [118, 144, 46].
- The assumption of autoscaling bundles (e.g., the VM instances from Amazon EC2) has been made in a considerable amount of SSCAS studies. However, it is known that renting bundles cannot and does not reflect the interests of consumers and the actual demand of their cloud-based services [78]. Thus, considering arbitrary and custom combinations of configurations and resources is an inevitable trend in the cloud computing paradigm.
- Considering multi-objectivity in SSCAS is a must to create better diversity and possibly better tradeoff quality without the needs of weights specification. In addition, how to achieve balanced tradeoff over the set of non-dominated solutions is worth investigating [49].
- More real-world cases and scenarios are needed, as this can be the only way to fully verify the potentials, effectiveness, and impacts of SSCAS.

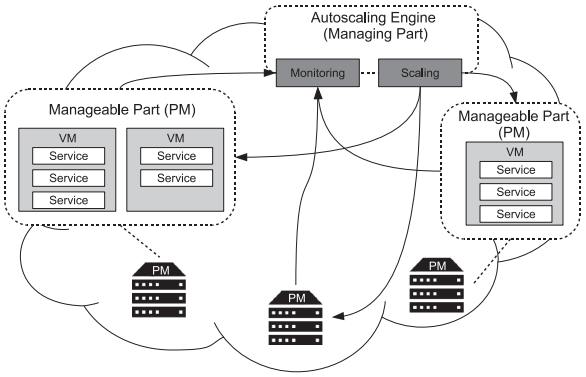


Fig. 1. The conceptual design of an autoscaling system. (Note that this figure represents the conceptual design of an autoscaling system in the cloud. Practical deployment of the autoscaling engine can be either centralized or decentralized where there are different engines, each of which is running on a PM.

This article is structured as follows: Section 2 introduces the background and challenges. Section 3 compares our work with the other reviews. A taxonomy and findings of the survey, with respect to different aspects of a SSCAS, are presented in Section 4. Section 5 discusses the findings and presents the open challenges we learned. Section 6 presents the conclusion.

2 BACKGOUND

2.1 Autoscaling System in Cloud

Given that it is almost impossible to access the low-level details of cloud-based services (e.g., their codes and algorithms) at runtime, an autoscaling system is often designed as two physical parts: a managing part containing the autoscaling engine and a manageable part encapsulating services and VMs/containers running in the cloud. The two physical parts are seamlessly and transparently connected for realizing the entire autoscaling process, known as *external adaptation* [56, 128].

The external adaptation of an autoscaling system is shown in Figure 1. As we can see, the core of an autoscaling system in the cloud is the autoscaling engine, which can consist of multiple logical aspects. A typical example of autoscaling system is a feedback loop that covers monitoring and scaling: The former gathers the service's or application's current state while the latter utilizes the information to decide an action after being analyzed and reasoned about by the autoscaling engine. Given the multi-tenant nature of the cloud, cloud-based services often come with different QoS objectives, SLA requirements and budget constraints, and so on. The ultimate goal of an autoscaling system is to adapt those cloud-based services, through scaling the related software configurations and hardware resources, in such a way that their objectives are continually optimized. To execute an autoscaling decision, the scaling actions could be vertical (scale up/down where changes occurs on a VM/container), horizontal (scale in/out that adds/removes other VMs/containers), or both.

2.2 Self-Adaptivity in Software Systems

The broad category of automatic and adaptive systems aim to deal with the dynamics that the system exhibited without human intervention, but this does not necessarily involve uncertainty, i.e., there are changes related to the system but it is easy to know when they would occur and the extent of these changes. Self-adaptivity, being a sub-category, is a particular capability of the system to handle both dynamics and uncertainty. Here self-adaptive systems refer to the systems that are capable of adapting their behaviors according to the perception of the uncertain environment and its own state. To date, self-adaptivity in software systems remains an important and

challenging research field [55, 92, 51]. According to the adaptive behaviors, self-adaptivity can be regarded as the following four properties, each of which covers a specific set of goals, as discussed by in Reference [128]:

- **Self-configuring:** "The capability of reconfiguring automatically and dynamically in response to change by installing, updating, integrating, and composing/decomposing software entities" [128].
- **Self-healing:** "This is the capability of discovering, diagnosing, and reacting to disruptions. It can also anticipate potential problems, and accordingly take proper actions to prevent a failure" [128].
- **Self-optimizing:** "This is also called self-tuning or self-adjusting, is the capability of managing performance and resource allocation in order to satisfy the requirements of different users, e.g., response time, throughput and utilization" [128].
- **Self-protecting:** "This is the capability of detecting security breaches and recovering from their effects. It has two aspects, namely defending the system against malicious attacks, and anticipating problems and taking actions to avoid them or to mitigate their effects" [128].

2.3 Self-Awareness in Software Systems

In contrast, self-awareness is about the capability of a system to acquire knowledge about its current state and the environment. Such knowledge permits better reasoning about the system's adaptive behaviors. Consequently, self-awareness is often seen as the lowest level of abstraction of self-adaptivity [128], and thus it can improve the basic perceptions and self-adaptivity of a system [54, 102, 101, 53]. Inspired from the psychology domain, Becker et al. [34] have classified self-awareness of a computing system into the following general capabilities (they have used node to represent any conceptual part of a system being managed):

- **Stimulus-aware:** "A node is stimulus-aware if it has knowledge of stimuli. The node is not able to distinguish between the sources of stimuli. It is a prerequisite for all other levels of self-awareness" [34].
- **Interaction-aware:** "A node is interaction-aware if it has knowledge that stimuli and its own actions form part of interactions with other nodes and the environment. It has knowledge via feedback loops that its actions can provoke, generate or cause specific reactions from the social or physical environment" [34].
- **Time-aware:** "A node is time-aware if it has knowledge of historical and/or likely future phenomena. Implementing time-awareness may involve the node possessing an explicit memory, capabilities of time series modeling and/or anticipation" [34].
- **Goal-aware:** "A node is goal-aware if it has knowledge of current goals, objectives, preferences and constraints. It is important to note that there is a difference between a goal existing implicitly in the design of a node, and the node having knowledge of that goal in such a way that it can reason about it. The former does not describe goal-awareness; the latter does" [34].
- **Meta-self-aware:** "A node is meta-self-aware if it has knowledge of its own capability(ies) of awareness and the degree of complexity with which the capability(ies) are exercised. Such awareness permits a node to reason about the benefits and costs of maintaining a certain capability of awareness" [34].

3 COMPARISON TO RELATED SURVEYS

Research on cloud autoscaling systems and the related topics have been reviewed in some other surveys. For example, Manvi and Shyam [113] present a review on resource management in the

cloud, particularly at the IaaS level. They have provided a board survey on different issues related to managing cloud resources, e.g., resource adaptation, resource mapping, resource brokering and so on. While resource management has some similarities to autoscaling, they lie in different levels of abstraction: The latter is more specific than the former. In other words, cloud autoscaling is one, but probably the most important, part of the board cloud resource management. Another review from Mana [112] is explicitly concerned with the VM-to-PM mapping problem that is also belongs to the cloud resource management category but is often regarded as a fundamentally different issue from cloud autoscaling. Ardagna et al. [27] present a survey on QoS modeling and its application in the cloud. Indeed, QoS is the major concern for a cloud autoscaling system, but its management can be governed by various different approaches other than autoscaling, e.g., load balancing and admission control, which are also covered by Ardagna et al. [27]. In contrast to the above, our survey has explicitly focused on automatically scaling software configuration and hardware provisioning in the cloud to change the capacity of cloud-based services to handle the dynamic workloads.

Al-Dhuraibi et al. [22] present a review on elastic autoscaling approaches in the cloud, specifically focusing on the physical infrastructure level support, e.g., benchmarking and containerization techniques. Our survey, in contrast, is primarily concerned with the logical architecture and algorithmic level techniques for achieving different aspects of cloud autoscaling, e.g., modeling and decision making. Qu et al. [122] also survey autoscaling approaches for a special type of cloud application, i.e., web applications, with a coarse correlation to self-adaptivity, e.g., if an approach is self-adaptive or not; while our survey is application agnostic and we present a finer correlation of an approach to different levels of self-adaptivity, e.g., self-optimization. The most related survey from the literature is probably the one by Lorido-Botran et al. [109], in which different category of algorithmic level techniques for QoS modeling and decision making in cloud autoscaling are reviewed. However, their survey differs from ours in the following three aspects: (i) they have not provided a comprehensive taxonomy on the cloud autoscaling problem; such a taxonomy (i.e., modeling, architecture, granularity, and decision making), which we will present in the next section, is important, as it clearly state the open problems and challenges related to different aspects of the cloud autoscaling domain, providing better clarifications and clearer directions for researchers on this research field. (ii) In addition, Lorido-Botran et al. [109] did not explicitly link the cloud autoscaling systems to different levels of self-awareness and self-adaptivity, which is one of the key contributions of our survey. (iii) Finally, we discuss the open problems and challenges of cloud autoscaling systems in a broader fashion.

In summary, our survey differs from the other similar reviews in the following ways:

- We present a focused survey on the logical architecture and algorithmic level techniques for cloud autoscaling that are application agnostic.
- We explicitly correlate the reviewed approaches with different levels of self-awareness, self-adaptivity, and the required knowledge in a fine-grained manner.
- We provide a clarified taxonomy that covers different logical aspects for engineering cloud autoscaling systems and classify every study accordingly.
- We discuss the open problems and challenges of cloud autoscaling systems in a broader fashion.

4 TAXONOMY AND SURVEY RESULTS FOR SSCAS

In this section, we present a taxonomy and survey results for the state-of-the-art SSCAS research obtained from our review process.

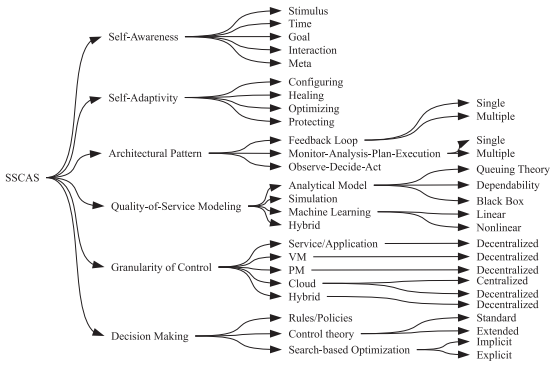


Fig. 2. A taxonomy of SSCAS research.

4.1 Review Process and Research Questions

The review is intended to create a broad scope to cover the landscape of SSCAS research. In articular, the following research questions serve as the main drivers of this review:

- *RQ1:* What are the levels of self-awareness and self-adaptivity that have been captured in SSCAS?
- *RQ2:* What are the architectural patterns used for engineering SSCAS?
- *RQ3:* What are the approaches used to model the quality related to SSCAS?
- *RQ4:* What is the granularity of control in SSCAS?
- *RQ5:* What are the approaches used for decision making in SSCAS?

The following prominent indexing services were used during the review: IEEE Xplore, ACM Digital Library, Science Direct, ISI Web of Knowledge, and Google Scholar. The search term was "Cloud computing" AND "Autoscaling" AND ("QoS modeling" OR "Performance modeling" OR "decision making" OR "optimization" OR "Architecture" OR "Interference" OR "Resource allocation"). After applying inclusion (e.g., considering only journal, conference, and workshop articles) and exclusion criteria (e.g., removing duplicate entries and considering only the extended version) to the initial search result, the review has ended with the total of 109 studies.

4.2 A Taxonomy of SSCAS Research

The overall taxonomy, concluded from the extracted studies, is given in Figure 2. As we can see, current research on SSCAS often require sophisticated designs in different highest-level logical aspects of the autoscaling engine, which we have classified and discussed as the following:

— ***Self-Awareness:*** This is concerned with the ability to acquire and maintain knowledge about the system's own states and the environment, as specified in Section 2.2. The key challenges here are as follows: Which level(s) of knowledge is required for a SSCAS?, What does it means for a certain level in the problem context (e.g., what does interaction refers to?), and What is the representation for different levels of knowledge, e.g., how do we represent goals in the SSCAS?

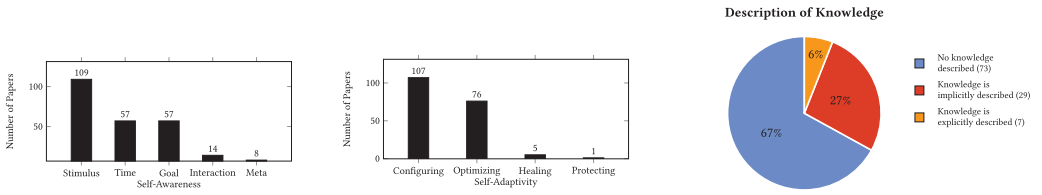
— ***Self-Adaptivity:*** This is about the ability to change the system's own behavior with specific goals in mind, as specified in Section 2.3. Often, the required levels of self-adaptivity depending on the requirements, but they could be also related to the specific levels of knowledge that the SSCAS is able to capture, e.g., the SSCAS has to be goal-aware to achieve self-optimization.

— ***Architectural Pattern:*** Autoscaling architecture is the most essential element of SSCAS. It describes the structure of the autoscaling process, the interaction between components and the modularization of the other important logical aspects in autoscaling. The challenge of architecting SSCAS is concerned with how to systematically capture different logical aspects (e.g., decision making) of SSCAS using a given architectural pattern. More importantly, how to encapsulate these aspects and the algorithms that realizes them into different components of the pattern.

— ***QoS Modeling:*** While modeling the cost incurred by cloud-based services is straightforward, modeling the QoS is often much more complex and challenging. Here the QoS modeling is concerned with the sensitivity of QoS with respect to the environment conditions (e.g., workload) and the control knobs (e.g., software configurations and hardware resources). The resulting model is a powerful tool to assist the autoscaling decision-making process. Without loss of generality, in this article, we use ***cloud primitives*** to refer to both control knobs and environmental conditions in the cloud. In particular, we further decompose the notion of primitives into two categories, termed ***control primitive*** and ***environmental primitive***. Control primitives refer to the internal control knobs that can be either software or hardware. They are the fundamental features that can be controlled by the cloud providers to support QoS. Specifically, software control primitives are the key cloud configurations at the software level, e.g., the number of threads in the thread pool, the buffer size, and the cache size, and so on. In contrast, hardware control primitives refer to the computational resources, such as CPU, memory and bandwidth, and so on. Typically, software control primitives exist on the PaaS layer while the hardware control primitives present on the IaaS layer. It is worth noting that considering software control primitives when autoscaling in the cloud is a non-trivial task, as they have been proved to be important features that can significantly influence the QoS [39, 152, 111]. The environmental primitives, on the other hand, refer to those external stimuli that is uncontrollable but can cause dynamics and uncertainties in the cloud. These, for example, can be the workload, incoming data, node failure, and so on. The above examples of primitives listed above are not exhaustive; Ghanbari et al. [78] have provided a more completed and detailed list of the possible control primitives in cloud.

  The challenges of QoS modeling include the following: (i) which primitives should be selected as model's input features; (ii) how does the QoS change in conjunction with those primitives; (iii) how to incorporate the information of QoS interference into the model; (iv) whether the model is built offline or online; (v) and whether the model is dynamic, semidynamic, or static.

— ***Granularity of control:*** Determining the granularity of control in the autoscaling engine is essential to ensure the benefits (e.g., QoS and cost objectives) for all cloud-based service. It is concerned with understanding whether certain objectives can be considered in isolation with some of the others, i.e., the boundary of decision making. This is because ***objective-dependency*** (i.e., conflicted or harmonic objectives) often exists in the decisions-making process, which implies that the overall quality of autoscaling can be significantly affected by the inclusion of conflicted or harmonic objectives when making a decision, hence rendering it as a complex task. This is especially important for the shared cloud infrastructure where objective-dependency exists for both intra- and inter-services. That is to say, objective-dependency is not only caused by the nature of objectives (intra-service), e.g., throughput and cost objective of a service; but also by the QoS interference (inter-services) due to the co-located services on a VM/container and co-hosted VMs/containers on a PM [39, 152, 111, 127].



(a) Inclusion of self-awareness levels (b) Inclusion of self-adaptivity levels (c) Discussion of required knowledge

Fig. 3. Paper count on self-awareness, self-adaptivity levels and how knowledge is discussed on the SSCAS architecture.

Here, the challenges are which granularity of control to use, what is the basic entity to control (e.g., application or VM), and whether the control is in a centralized or decentralized manner.

— ***Decision making:*** The final logical aspect in autoscaling logic is the dynamic decision-making process that produces the optimal (or near-optimal) decision, which consists of the newly configured values of the related control primitives, for all the related objectives. In the presence of objective-dependency, autoscaling decision making requires us to resolve complex tradeoffs, subject to the SLA and budget requirements. The tradeoff decision can be then executed using either vertical (scale up/down) and/or horizontal scaling actions (scale in/out), which adapt the cloud-based services and/or VMs/containers correspondingly.

The challenges of decision making in SSCAS include the following: (i) how to reason about and search for the effected adaptation decisions; (ii) what are the objectives and their representations and conflicting relations, if any; and (iii) which are the control primitives to tune.

In the following, we present our detailed findings in regards to the taxonomy of SSCAS.

4.3 The Levels of Self-Awareness and Self-Adaptivity in Cloud Autoscaling Systems

RQ1: What are the levels of self-awareness and self-adaptivity that have been captured in SSCAS?

It is worth noting that not all the studies have explicitly declared which levels of self-awareness/self-adaptivity that they have taken into account, therefore we identified this by looking at the studies in details with respect to the definitions of self-awareness/self-adaptivity. From Figure 3(a), we can see that *stimulus-awareness*, which is the fundamental level in self-awareness, has been considered in all the studies. The *time-* and *goal-awareness* have attracted relativity similar amount of attention. However, *interaction-* and *meta-awareness* has not been widely studied in recent SSCAS research. In Figure 3(b), we see that *self-configuring* and *-optimizing* have been predominately captured in the studies, whereas *self-healing* and *-protecting* receive little attention. In particular, we found only one study that explicitly aims for *self-protecting* in SSCAS. Figure 3(c) illustrates whether the required knowledge representations at the SSCAS architecture level, e.g., knowledge of goal is required in the architecture, have been discussed, implicitly discussed or explicitly discussed in the studies. We can see that the majority of the studies surveyed do not attempt to declare what knowledge is required in SSCAS architecture, leaving only 33% of the studies have discussed the knowledge implicitly or explicitly.

4.4 Architectural Pattern

RQ2: What are the architectural patterns used for engineering SSCAS?

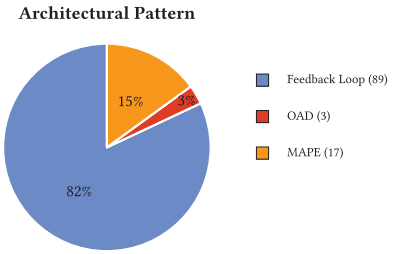


Fig. 4. Number of articles per architectural pattern.

Table 1. Detailed Classifications of Architectural Patterns for SSCAS

Architectural Pattern	***Style***	***Open/Close***	***Representative Examples***
Feedback loop	Single	Close	[70, 105, 144, 121, 64, 88, 120, 75, 89, 69, 132, 74, 60, 80, 26, 82, 107, 24, 71, 72, 40, 142, 93, 155, 149, 125, 138, 103, 139, 57, 62, 61, 127, 141, 111, 98, 36, 145, 153, 77, 129, 148, 83, 126, 25, 42, 90, 130, 133, 68, 131, 76, 123, 134, 63, 124, 33, 150, 106, 135, 32]
		Open	[73, 151, 91]
	Multiple	Close	[92, 28, 29, 118, 143, 146, 94, 59, 65, 140, 147, 39, 154, 50, 49, 47, 81, 21, 119, 96, 45, 97, 52, 48, 110]
OAD	Single	Close	[136, 84, 86]
MAPE	Single	Close	[104, 43, 37, 114, 41, 100, 35, 67, 66, 115, 116, 99, 31, 79]
	Multiple	Close	[152, 20, 46]

We classified the predominantly applied architectural patterns for SSCAS into three categories based on their basic form; these are *Feedback Loop* [38], *Observe-Decide-Act* (ODA) [85], and *Monitor-Analysis-Plan-Execute* (MAPE) [87].

From Figure 4, we see that the generic feedback loop has been the predominant architecture pattern in SSCAS, following by the MAPE pattern. Particularly, as we can observe from Table 1, single and close feedback loop are widely exploited in SSCAS. In the following, we specify some representative studies under each category in details.

4.4.1 Feedback Loop. Feedback loop is the most general architectural pattern for controlling self-adaptive systems, including the autoscaling systems. It is usually a closed-form loop made up of the managing system itself and the path transmitting its origin (e.g., a sensor) to its destination (e.g., an actuator). Here we further divide the pattern as *single* or *multiple* loops:

- *4.4.1.1 Single Loop:* Single loop is the simplest, yet the most commonly used, pattern for SSCAS due to its flexibility. The most common practice with single loop is to build a closed feedback control where the core is the decision-making component and an optional QoS modeling component, e.g., Ferretti et al. [70], CloudOpt [105], SmartSLA [144], CLOUDFARM [120], and so on. Some other studies have included an additional component for workload or demand prediction based on either offline profiling, e.g., Jiang et al. [89] and Fernandez et al. [69], or online learning, e.g., Kingfisher [132] and PRESS [80].

Open feedback exists for single loop, as presented in Cloudine [73], where the scaling actions are partially triggered by user requests. In particular, they use a centralized *Resource and Execution Manager* to handle all the scaling actions. Apart from the general autoscaling architecture, other efforts are particularly designed upon specific cloud providers. For example, Zhang et al. [151] as well as Kabir and Chiu [91] propose to use a simple feedback loop for architecting autoscaling system, which is heavily tied to the properties of Amazon EC2.

- *4.4.1.2 Multiple Loops:* It is possible to use multiple loops and controllers for autoscaling in the cloud. Here multiple feedback loops operate in different levels of the architecture, e.g., one operates at the cloud level while the others operate on each VM. The benefit is that multiple loops provide low coupling in the design of the loops for SSCAS. Notably, multiple loop control can be used to separate global and local controls. Among others, Kalyvianaki et al. [92] apply multiple decentralized feedback loops for autoscaling CPU in the cloud. Although it aims to exploit one loop per individual application, the controllers actually operate on each tier of an application. Chen and Bahsoon [49] also leverage multiple feedback loop to auto-scale cloud services, where each PM maintains a loop. Unlike classic feedback loop where the adaptations occur only on the manageable part of SSCAS, their adaptations also happen on the managing part.

  Multiple loop control is also effective for isolating the logical aspects of autoscaling and management in the cloud. For instance, Wang et al. [140] propose a two-layered feedback control for autoscaling in the cloud. The first layer, termed guest-to-host optimization, controls the hardware resources, e.g., CPU and memory. Subsequently, the host-to-guest optimization adapts the software configuration accordingly.

4.4.2 Observe-Decide-Act. Observe-Decide-Act (ODA) loop [85] is considered as an extended pattern of the generic feedback loop. As specified in the SEEC framework [85], ODA is unique in the sense that it decouples multiple loops to different roles (i.e., application developer, system developer, and the SEEC runtime decision infrastructure) in the development lifecycle, each role focuses on one or more steps in ODA. In such a way, ODA links the effects of human activities on the adaptive behaviors.

Among others application of OAD in SSCAS, MNEMOS [136] has relied on OAD to realize an integrated, datacenter-wide architecture for autoscaling resources in the cloud, in which the *System Monitor* acts as the observer, the *Portfolio Scheduler* acts as the decider, and the *VM Manager* acts as the executioner. Huber et al. [86] also use ODA for self-aware autoscaling resources in the cloud. However, unlike a traditional ODA loop, it has an additional *Analysis* step that is used to detect the type of problems that trigger adaptation.

4.4.3 Monitor-Analyze-Plan-Execute. Another pattern extended from the generic feedback loop, namely Monitor-Analyze-Plan-Execute (MAPE), is first proposed by IBM for architecting self-adaptive systems. In such a pattern, the *Decide* step in OAD is further divided into two substeps, *Analyze* and *Plan*, where the former is particularly designed to determine the causes for adaptations, e.g., SLA violation; the latter, however, is responsible for reasoning about the possible actions for adaptation. MAPE sometime can be extended by a Knowledge component (MAPE-K) that maintains historical data and knowledge used by the system for better adaptation. MAPE can be also realized as either *single* or *multiple* loops:

- *4.4.3.1 Single Loop:* MAPE (or MAPE-K) is also widely applied for SSCAS. For example, the architecture of the FoSII project [37, 114] leverages single MAPE-K to realize the self-management interface, aiming to prevent SLA violations in cloud by devising the related actions. They also use the additional *Knowledge* (K) component to record cases

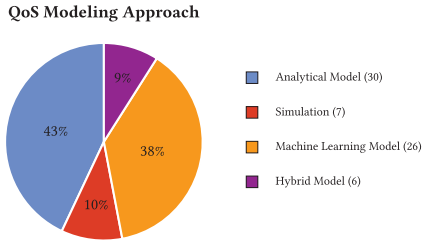


Fig. 5. Number of articles per QoS modeling approach.

and the related solutions, which can assist the autoscaling decision making. Chen and Bahsoon [46] have realized MAPE as a single loop where the adaptations occur on both the managing and manageable parts of the SSCAS.

- *4.4.3.2 Multiple Loops:* Realizing multiple MAPE loops for SSCAS is also possible. Zhang et al. [152] introduce an architecture for autoscaling using two nested MAPE loops. The first loop is responsible for adapting the software primitives while the other loop is used to change the hardware primitives. These two loops run sequentially upon autoscaling, that is, adapting the software control primitives before changing the hardware control primitives. Similarly, BRGA [20] utilizes MAPE to realize a framework for autoscaling in the cloud. Such solution consists of both the local and global view of the cloud-based application.

4.5 QoS Modeling

RQ3: What are the approaches used to model the quality related to SSCAS?

QoS modeling, or performance modeling, is a fundamental research theme in cloud computing, and it can serve as useful foundations for addressing many research problems in the cloud [109], including autoscaling. The QoS models correlate the QoS attributes to various control primitives and environmental primitives. Clearly, these models are particularly important for SSCAS, since they are powerful tools that can assist the reasoning about the effects of adaptation on objectives in the autoscaling decision-making process. Note that although QoS model can provide great help to the decision making in SSCAS, not all of the studies have considered QoS modeling as part of their solutions. In fact, some of them rely on model-free solutions, e.g., control-theoretic approaches, which we will review in Section 4.7.

Typically, QoS modeling consists of two phases: the primitives selection phase and the QoS model construction phase. More precisely, the primitives selection phase determines *which* and *when* the cloud primitives correlate with the QoS, while the QoS model construction phase identifies *how* these primitives correlate with the QoS, i.e., their magnitudes in the correlation. The QoS models might come as three forms as follows: (i) static models where the models' expression and their structure (e.g., the number of inputs and the coefficients) do not change over time; (ii) dynamic models that permit those changes; or (iii) semi-dynamic models in the sense that the expression (e.g., coefficients) could be dynamically updated but the input features do not. Further, those models can be built online at system runtime or offline at the design phase of the system. In this section, we classified the studies mainly based on the modeling methods applied to the QoS model construction phase, since we found that the primitives selection is often conducted using manual and static approaches in the studies. As we can see in Figure 5, the majority of the studies has exploited analytical models (43%) and machine-learning models (38%) to predict QoS. In contrast, simulation and hybrid models receive much less attention. From Table 2, we can obtain the following observations:

Table 2. Detailed Classifications of QoS Modeling Approaches for SSCAS

Modeling Approach	*Type*	*Built*	*QoS Interference*	*State*	*Concrete Models*
Analytical model	Queuing model	Offline	No	Static	**S-QUEUE (5):** [151, 88, 138, 139, 89] **M-QUEUES (3):** [81, 35, 103] **LQN (4):** [57, 105, 104, 155]
	Dependability model	Online	No	Semi	**MDP (1):** [41] **MODEL@RUNTIME (1):** [64]
		Offline	No	Static	**PCM (1):** [86] **GRAPH (1):** [62]
		Offline	No	Dynamic	**PCA (1):** [94]
	Black box model	Offline	No	Static	**EMPIRICAL-MODEL (9):** [65, 37, 120, 67, 66, 21, 91, 20, 131, 116, 79, 32]
		Offline	Yes	Static	**EMPIRICAL-MODEL (1):** [110]
Simulation		Offline	No	Semi	**PROFILING (2):** [69, 134] **SIMULATOR (4):** [71, 40, 142, 93, 72]
Machine-learning model	Linear	Online	No	Semi	**LR (3):** [107, 152, 61] **ARMA (1):** [121, 118] **KF (1):** [92]
		Online	No	Dynamic	**LR (1):** [95]
		Online	Yes	Semi	**MIMO (4):** [100, 127, 141, 111]
	Nonlinear	Online	No	Semi	**KM (1):** [74] **RT (1):** [144] **ANN (4):** [98, 119, 96, 45] **SVM (2):** [59, 98] **CHANGE-POINT (1):** [36]
		Online	Yes	Semi	**SVM (1):** [124]
	Ensemble	Online	No	Semi	**ARMA+SVM (1):** [154] **ANN+ANN (1):** [97]
		Online	No	Dynamic	**KNN+LR:+RT (1):** [145]
		Online	Yes	Dynamic	**ARMA+ANN+RT (2):** [52, 48]
Hybrid model		Semi	No	Semi	**LQN+KF (2):** [75, 153] **LQN+KF+K-MEAN (1):** [77] **S-QUEUE+ARMA (1):** [146] **M-QUEUE+KF (1):** [76]
		Offline	Yes	Static	**EMPIRICAL-MODEL+PROFILING (1):** [135]

Abbreviations: S-QUEUE = Single Queue; M-QUEUE = Multiple Queues; LQN = Layered Queuing Network; MDP = Markov Decision Process; PCM = Palladio Component Model; PCA = Principal Component Analysis; LR = Linear Regression; ARMA = Auto-Regressive Moving-Average; KF = Kalman Filter; MIMO = Multiple-Input and Multiple-Output; KM = Kriging Model; RT = Regression Tree; ANN = Artificial Neural Network; SVM = Support Vector Machine; KNN = k-Nearest-Neighbor.

(1) Despite the high importance of QoS interference, it has not received much attention when modeling the QoS (only 13%).
(2) Truly dynamic QoS modeling, i.e., changing both the input features and their coefficients, is still minority (7%). Other studies have merely considered changing coefficients of the model while ignoring the input features' dynamics (54% semi-dynamic) or none at all (39% static).
(3) The concrete modeling methods applied for machine-learning models is more diverse than the methods in other categories.

Additionally, from Table 3, we can observe that

(1) Although most studies (65%) have only considered certain inputs/outputs during their experiments, they have claimed that their model is compatible with any given inputs (i.e., any cloud control primitives) and/or output (i.e., any QoS attributes).
(2) The most widely considered input dimension is CPU while the most common output is response time (except the general one, i.e., QoS attribute).
(3) The most explicitly modeled number of outputs is three, while the most explicitly considered number of inputs is four.

In the following, we specify some representative studies on QoS modeling for SSCAS in detail.

4.5.1 Analytical Modeling. Analytical modeling approaches rely on a closed-form structure to model the cloud-based service. These models are often built offline based on theoretical principles and assumptions. Next, we further divide the analytical modeling approach into *queuing theory*, *dependability models*, and *black box models*.

- *4.5.1.1 Queuing theory:* Queuing model and queuing network are widely applied for QoS modeling in the cloud. They model the cloud-based services as a single queue or a collection of queues that are interacting through a mixture of request arrivals and completes. Specifically, a single queue has been used to model the correlation of response time (or throughput) to CPU, number of VM, and workload. For example, depending on the assumption of the distribution on arrival and service rate, the model can be built as M/G/m queue[3] by Zhang et al. [151], M/G/m queue by Jiang et al. [88], M/M/1 queue by E^3-R [138] and JustSAT [139], and M/M/m queue by Jiang et al. [89]. To create more detailed modeling with respect to the internal structure of cloud-based services, multiple queues can be used to create QoS models; for example, Goudarzi and Pedram [81] apply multiple queues to model the response time for cloud-based multi-tiered applications with respect to number of VM and workload. Their work calculates average response time for the queue in the forward direction throughout the tiers.

  Unlike classical queuing model and queuing network, the Layer Queuing Network (LQN) additionally models the dependencies presented in a complex workflow of requests to cloud-based services and applications. For instance, Zhu et al. [155] have also used LQN where the authors employ a global M/M/m queue for the entire on-demand dispatcher and then a M/G/1 queue on each tier of an application. The former queue correlates the response time to the number of VMs, while the latter queue models the relationship between response time and CPU of the VM that contains the corresponding tier.
- *4.5.1.2 Dependability models:* Dependability models focus on the modeling of various states for QoS attributes. For example, in QoSMOS [41], the authors analytically solve the Markov Models (Discrete-Time Markov Chain and Markov Decision Process) to model the QoS for services in an application. The model correlates QoS attributes with hardware resources and workload. Huber et al. [86] uses Palladio Component Model (PCM) as architecture-level QoS model, since it permits us to explicitly model different usage profiles and resource allocations.
- *4.5.1.3 Black box models:* Black box models handle the QoS using empirical and historical domain knowledge. Among others, the CLOUDFARM framework [120] uses an empirical QoS model where the correlation between certain QoS values and the required resource is captured (i.e., CPU). In particular, the authors assumed that the magnitudes of resources to

[3]In queuing theory, M denote Poisson distribution and G denotes arbitrary distribution. A term $M/G/m$ refers to Poisson distribution of arrival rate and arbitrary distribution of service rate, and there exists m servers.

Table 3. QoS Modeling Approaches for SSCAS by Inputs and Output

Outputs	*Cloud Primitives*
QoS attributes	**CPU (1):** [86]
	Number of VM (1): [64, 69]
	Configurations (1): [59, 111]
	Resources (2): [119, 127]
	CPU and bandwidth (1): [141]
	CPU and memory (4): [74, 118, 100, 72]
	Configurations and resources (6): [94, 152, 41, 52, 48, 45]
	Configurations, CPU and memory (1): [154]
	Resources and workload (8): [95, 96, 153, 71, 40, 142, 93, 79]
	CPU, memory and disk (2): [62, 121]
	CPU, memory and bandwidth (3): [120, 98, 115]
	CPU, storage and bandwidth (2): [65, 67]
	Configurations, resources and workload (1): [145]
	CPU, bandwidth, storage and number of VM (1): [37]
	CPU, memory, workload and number of VM (1): [144]
	CPU, memory, workload and bandwidth: (1) [61]
	CPU, memory, storage and bandwidth (1): [66]
	CPU, number of VMs and workload (1): [110]
	Workload and interference index (1): [124]
Response time	**CPU (1):** [92]
	Number of VM (1): [81]
	Workload and number of VMs (4): [57, 35, 139, 155]
	CPU and memory (2): [138, 151]
	Workload and CPU (2): [104, 146]
	Thread and CPU (1): [103]
	CPU, memory, workload and number of VMs (2): [105, 75]
	CPU, workload and number of VMs (1): [76]
Response time and workload	**Number of VMs (1):** [91]
	CPU and memory (1): [21]
	Workload and number of VMs (1): [89]
	Workload, number of VMs and VM type (1): [135]
Response time and utilization	**Number of VMs and VM type (1):** [32]
Response time and throughput	**CPU and memory (1):** [88]
	CPU, memory, number of VMs and VM type (1): [134]
CPU utilization	**Workload and number of VM (1):** [107]
QoS attributes and hardware demand	**Configurations and resources (1):** [97]
	CPU and workload (1): [77]
QoS attributes and workload	**Workload and number of PM (1):** [36]
QoS attributes and overhead	**Resources (1):** [20]
Cost	**Workload and number of VM (2):** [131, 116]

the QoS values is known, as specified by the cloud service or application provider. Another study from Emeakaroha et al. [67, 66] proposes an empirical model that maps the expected QoS values with CPU, memory, bandwidth, and storage based on the assumptions of the system that being managed.

4.5.2 Simulation-Based Modeling. QoS models can be also generated by various simulators; here, conducting simulations is usually a complex and expensive process and thus they are used in an offline manner. In practice, simulation is required to be setup by the domain experts, who will often need to analyze, interpret, and profile the data collected after simulation runs. Specifically, Fernandez et al. [69] have relied on a profiling approach that builds the QoS model for each bundle of VM offline. The process is similar to a simulation modeling approach. CDOSim [71] is a framework that simulates the actual application in the cloud to restrict the search-space for autoscaling and to steer the exploration towards promising decisions. CloudSim [40] is a simulation toolkit that models QoS attributes (of VM) with respect to resource allocation. It supports both single-cloud and multiple-cloud scenarios. As an extension of CloudSim, CloudAnalyst [142] allows the simulation of QoS attributes for the application deployed on geographically distributed datacenters. Similarly, DCSim [93] simulates the overall quality of resource autoscaling for the entire cloud.

4.5.3 Machine-Learning-Based Modeling. The increasing complexity of cloud-based services has rendered the modeling process an extremely difficult task for human experts. To this end, recent studies have exploited the advances of machine-learning algorithms and theory to create more reliable QoS models. In the following, we survey the key studies that apply machine-learning approaches for QoS modeling in the cloud. In particular, we have further classified them into two categories, these are *linear* and *nonlinear* modeling.

- *4.5.3.1 Linear modeling:* Learning algorithms based on linear models for QoS modeling in the cloud can handle linear correlation between a selected set of cloud primitives (e.g., CPU, memory, number of VM, workload, etc.) and output (i.e., QoS attributes), and they are sometimes very efficient. Simple linear models most commonly rely on linear regression, where each primitive input is associated with a time-varying weight, e.g., Lim et al. [107], Zhang et al. [152], and Collazo-Mojica et al. [61]. More advanced forms exist, e.g., Padala et al. [121], that have used an Auto-Regressive-Moving-Average (ARMA) model that is trained continually by Recursive Least Squares (RLS). The authors claim that the second-order linear ARMA model is easy to be estimated online and can simplify the corresponding controller design problem with adequate accuracy.

  We found that there are limited studies that attempt to capture the information of QoS interference in the linear QoS model, and they only focus on the VM-level [100, 127, 141, 111]. As an example, Q-Cloud [127] has explicitly considered QoS interference by using the hardware control primitives of all co-hosted VMs as inputs, rendering it in a Multi-Inputs-Multi-Output (MIMO) model, which is trained by the Least Mean Square (LMS) method.
- *4.5.3.2 Nonlinear modeling:* Learning algorithms based on nonlinear models for QoS modeling in the cloud is able to capture complex and nonlinear correlation, in addition to the linear one. However, it can also produce relatively large overhead than the linear modeling. Here, existing studies often aim to model the correlation between hardware control primitives (e.g., CPU, memory and bandwidth) and QoS. The nonlinear modeling can rely on the kriging model [74], Regression Tree (RT)[144], Artificial Neural Network (ANN) [98, 119, 96, 45], Support Vector Machine (SVM) [59, 98], and change-point detection [36]. For example, SmartSLA [144] employs Regression Tree (RT) and boosting to model the QoS.

RT partitions the parameter space in a top-down fashion and organizes the regions into a tree style. The tree is then trained by M5P where the leaves are regression models. The study from Kunda et al. [98] presents sub-modeling based on ANN and SVM for correlating QoS with hardware control primitives in the cloud. Instead of building a single model for a QoS attribute, they train n sub-models, whereby n is determined by performing k-means clustering based on the similarity between data values of QoS, creating more accurate and finer-grained models.

- *4.5.3.3 Ensemble modeling:* Examples exist for cases where multiple linear and/or nonlinear machine-learning algorithms are explored together. Among others, Chen et al. [52, 48] exploit a bucket of learning algorithms (both linear and non-linear models). The model accuracies are tracked continually at runtime, considering QoS interference. The best model for a given input values, according to both local and global errors, will be used to make a prediction.
- *4.5.3.4 Comparison of different learning algorithms:* Given the various types of machine-learning algorithms, it can be difficult to determine which one(s) are the appropriate algorithms for QoS modeling in the cloud with respect to both accuracy and overhead. There is research that have conducted empirical comparisons of different possible learning algorithms for QoS modeling in the cloud [117, 108, 58].

4.5.4 Hybrid Modeling. We discovered that linear machine-learning algorithms are also commonly used with analytical approaches to form QoS models. Specifically, Grandhi et al. [75] and Zheng et al. [153] have proposed a hybrid model: To model the multi-tiered application, they have relied on a modified LQN containing some time-varying coefficients. The authors then employ the Kalman filter as an online parameter estimator to continually estimate those coefficients. The approach proposed by Xiong et al. [146] has relied on a combined model, where an M/G/1 queue is used to model the correlation of workload to response time while ARMA is used to model the relationship between response time and CPU.

4.5.5 Dynamic Primitives Selection. We noticed that the majority of the above-mentioned studies regard the primitives selection as a manual and offline process; most commonly, they have relied on empirical knowledge and heavy human analysis to select the important primitives as the input features of QoS models. Although not many, there are some studies that explicitly consider dynamic process in primitives selection, which tends to be more accurate and can be easily applied [94, 95, 145, 48]. As an example, vPerfGuard [145] is a framework that correlates QoS attributes with respect to software control primitives, hardware control primitives, and environmental primitives. The authors achieve primitive selection based on both filter (relevance-based correlation coefficient) and wrapper (i.e., hill-climbing comparison for different learning algorithms). Chen and Bahsoon [48] dynamically select primitives that maximize both information relevance (between a primitive and quality) and minimize redundancy (between already selected primitives). While explicitly modeling the effects of QoS interference, the authors propose a fully self-adaptive approach that selects primitives that improve prediction accuracy given a learning algorithm.

4.5.6 Workload and Demand Modeling. We found that some existing studies (e.g., References [132, 129, 96]) attempts to model the workload and demand for assisting autoscaling decision making. In those cases, the modeling is reduced to a single dimension, where the core is to model the trend of the workload or demand using its historical data. However, unlike QoS modeling, which is often multi-dimensional, the single dimension in workload or demand models does not offer the ability to reason about the effects of autoscaling decisions and the possible tradeoffs.

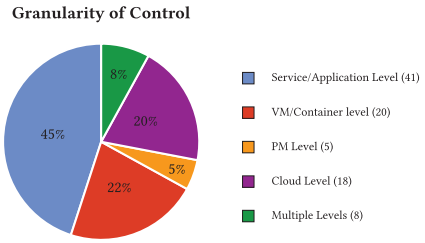


Fig. 6. Number of articles per granularity of control.

Table 4. The Granularity of Control in SSCAS

Granularity	*Entity*	*Style*	*Representative Examples*
Service & Application level	Application	Decentralized	[132, 139, 69, 61, 154, 103, 88, 43, 80, 94, 107, 129, 89, 81, 111, 30, 83, 75, 146, 126, 24, 121, 133, 68, 131, 116, 99, 31]
	Service	Decentralized	[62, 148, 37, 114, 74, 41, 64, 138, 72, 25, 42, 91, 86]
VM & Container level	Application	Decentralized	[140, 149, 82, 152, 59, 141, 127, 76],
	Application	Centralized	[79, 135, 32]
	VM	Decentralized	[26, 92, 90]
	VM	Centralized	[123, 63, 124]
	Container	Decentralized	[33, 106]
	Container	Centralized	[150]
PM level	Application	Decentralized	[125, 147, 39, 119, 100]
Cloud level	Application	Centralized	[57, 28, 73, 21, 120, 104, 155, 65, 105, 151, 29, 20, 115, 136, 84, 60]
	Application	Decentralized	[70, 143]
Hybrid levels	Application	Decentralized	[118, 144, 110]
	Application	Centralized	[130]
	Service	Decentralized	[50, 46, 49, 47]

4.6 Granularity of Control

RQ4: What is the granularity of control in SSCAS?

The ultimate goal of autoscaling is to optimize the QoS and cost objectives, which are referred to as *benefits*, for all cloud-based services. To this end, the granularity of control in autoscaling plays an integral role, since it determines the boundary of decision making: which and how many objectives should be considered in a decision-making process of autoscaling. In the following, we classify existing SSCAS studies depending on what level of granularity they operate.

As we can see from Figure 6, the predominant granularity of control is at the service/application level where the boundary of decisions making is grouped by each service/application. Notably, controlling at the cloud level tends to be the second most popular, leaving the other levels being a minority. Generally, the finer granularity of control implies that it is harder to achieve globally optimal benefits but likely to generate smaller overhead. However, globally optimal benefits can be reached more easily with large overhead if the granularity of control is coarser. According to Table 4, we can obtained the following observations:

(1) Most of the studies (72%) see each application as the basic entity regardless to the granularity of control in SSCAS.
(2) Decentralization (74%) is the most popular approach for all granularity of control, except the cloud level, where centralized (or partially centralized) control is predominately exploited.

In the following, we specify each granularity of control for SSCAS in detail.

4.6.1 Controlling at Service and Application Level. Service/Application level is the finest level of control in a SSCAS. It is worth noting that by service, we refer to any conceptual part of the system being managed. As a result, control granularity at the service/application level may refer to independently controlling/scaling an application, a tier of an application or a cloud-based service.

We found that most of the studies have focused on controlling each cloud-based application. These approaches have relied on controlling the QoS and/or cost for each individual application in isolation, and therefore, they sometimes regard an application as a service. Examples of such include the following: Lim et al. [107] control the application and its required VM, in which case an application is regarded as a service. Sedaghat et al. [129] regard application as a service and considered the required number of VMs and the fixed VM bundles for such service.

There are studies that explicitly controls cloud-based service in general. Among others, Copli et al. [62] control the QoS, cost, and their elasticity for each service deployed in the cloud. Yang et al. [148] control the cost of individual cloud-based services. The FoSII project [37] controls individual cloud-based service, their QoS, and cost. Gambi et al. [74] control at the service level, where the controller decides on the optimal autoscaling decision for cloud-based service in isolation.

4.6.2 Controlling at Virtual Machine and Container Level. VM hypervisor and container are two fundamental infrastructures that underpin cloud computing. In particular, VM and container differ in the sense that the former requires a full Operating System to be installed on a VM while the latter does not. This fact allows the container to set naively with the host PM, providing much faster creation and removal time of VM image. However, such benefit comes at the expense of weaker security guarantees and potentially greater chances of interference, given that the container instances have less isolation. Despite such a difference, the two infrastructures are conceptually similar, as they both aim to provide a certain level of isolation on top of the hosting PM, and thus they can be regarded as the same granularity of control.

VM level means that the control and decision making operate at each VM in the SSCAS. In particular, certain studies assumes a one-to-one mapping between application (or a tier) and VM, and thus they can be categorized as either service-level or VM-level granularity. To better separate them from the pure service/application–level granularity of control, these studies are regarded as VM-level granularity. Specifically, FC2Q [26] regards application tier and VM interchangeably; therefore, controlling each tier of an application is equivalent to control each individual VM. Similarly, Kalyvianaki et al. [92] control a tier of an application that resides on a VM, and the authors only focus on CPU allocation of a VM.

4.6.3 Controlling at Physical Machine Level. Autoscaling decision making on each PM independently is referred to as PM level control in the SSCAS. The primary intention of PM level control is to manage the QoS interference caused by co-hosted VMs. Among the others, Xu et al. [125, 147] control the VMs collectively at the PM level, and thus the autoscaling promotes better management of QoS interference at the VM level.

4.6.4 Controlling at Cloud Level. The most coarse level of control granularity is at the cloud level for SSCAS. The majority of the studies achieves autoscaling at the cloud level by using a

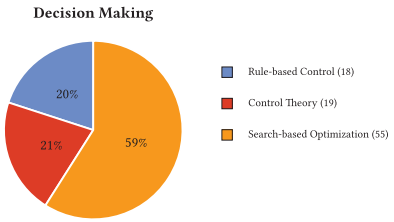


Fig. 7. Number of articles per decision-making approach.

centralized and global controller, with an aim to manage utility [57, 28, 44, 73, 21, 120], profits [104, 155], and availability [65]. Among others, Ferretti et al. [70] control the QoS for all cloud-based services in a global manner. However, the actual deployment can be either centralized or decentralized. Similarly, CRAMP [29] uses a centralized and global controller, and it controls the entire cloud for cost and QoS. CloudOpt [105] also controls the entire cloud using centralized control, as the considered optimization involves all the PM in the cloud.

Some of the studies have relied on a decentralized system where a consensus protocol is employed for controlling at the cloud granularity. For example, Wuhib et al. [143] aim to control the entire cloud, and thus the QoS and the overall power consumption of the cloud can be collectively managed. Further, they have relied on decentralized deployment, which can reduce the overhead of cloud-level control.

4.6.5 Controlling at Hybrid Level. We found that it is also possible for SSCAS to operate at multiple and hybrid levels, with an aim to better manage the overhead and global benefit. For example, Minarolli and Freisleben [118] combine both PM-level and cloud-level control, where the PM level is decentralized and the objective is to optimize the utility locally. Similarly, SmartSLA [144] aims to control the resource allocation for all the cloud-based services, and therefore it utilizes a global, cloud-level control in addition to the decentralized local control on each VM. Differently from the others, Chen and Bahsoon [46] exploit a dynamic schema where the multiple simultaneously presented granularity are changed at runtime, according to the objective-dependency.

4.7 Tradeoff Decision Making

RQ5: What are the approaches used for decision making in SSCAS?

The final important logical aspect in cloud autoscaling is the challenging decision-making process, with the goal to optimize QoS and cost objectives. It is even harder to handle the tradeoff between possibly conflicting objectives. Such a decision-making process is essentially a combinatorial optimization problem where the output is the optimal (or near-optimal) decision containing the newly configured values for all related control primitives. In the following, we survey the key studies on the decision making for SSCAS. In particular, we classify them into three categories; these are *rule-based control*, a *control-theoretic approach,* and *search-based optimization.*

As we can see from Figure 7, while rule-based control and the control-theoretic approach share similar popularity in the studies, search-based optimization receives much more attention than the other two. From Table 5, we can observe the following:

(1) Most of the studies (63%) in SSCAS do not attempt to consider the tradeoff between objectives during the decision making, or they handle such a tradeoff in the way that different objectives are aggregated using weighted sum (29%), which essentially converts the multiple objectives into a single one.
(2) Explicit consideration of QoS interference (16%) is still rare during the decision making.

Table 5. The Decision-Making Approaches for SSCAS

Decision-Making Approach	*Form*	*Tradeoff*	*QoS Interference*	*Concrete Methods*
Rule-based control		None	No	**RULES (17):** [73, 62, 70, 65, 143, 115, 83, 43, 136, 84, 130, 133, 31, 123, 79, 63, 106]
		None	Yes	**RULES (1):** [124]
Control theory	Standard	None	No	**PDC (3):** [28, 107, 29] **KC (2):** [92, 75] **FC (3):** [26, 24, 140]
		Weighted sum	No	**FC (1):** [126]
	Extended	None	No	**PIC+LA (1):** [146] **ANN+FC (1):** [82] **MPC+MA (1):** [68] **MIMO(1):** [99] **PIC+ILP (1):** [33]
		Weighted sum	No	**FC+QPS (1):** [118] **PIDC+RL+ES (1):** [154]
		Weighted sum	Yes	**MPC+QP (1):** [100]
		None	Yes	**FUZZY-MIMO (1):** [141] **MIMO (1):** [127]
Search-based optimization	Implicit	None	Yes	**RL (3):** [147, 125, 39]
		None	No	**RL (1):** [149] **DEMAND (5):** [25, 80, 42, 60, 90]
	Explicit	Single	No	**HEURISTIC (2):** [148, 110] **MIP (1):** [105] **DP (1):** [151] **ACO (1):** [30] **ES (7):** [94, 139, 61, 86, 131, 116, 150] **LA (1):** [59] **ILP (2):** [132, 129]
		Single	Yes	**HEURISTIC+DT (1):** [111]
		Weighted sum	Yes	**ES (2):** [57, 76] **RS (1):** [46] **HEURISTIC (1):** [135]
		Weighted sum	No	**ES (6):** [41, 74, 89, 37, 114, 32] **LS (1):** [21] **NFM (1):** [104] **GS (1):** [144] **FDS (1):** [81] **DT (1):** [69] **BS (1):** [91] **QP (1):** [121] **DP (1):** [120]
		Weighted sum	No	**TS (1):** [155] **GA (3):** [152, 119, 20] **PSO (1):** [152]
		Pareto	No	**SMS-MOEA (1):** [103] **NSGA-II (3):** [64, 138, 72]
		Pareto	Yes	**MOACO (3):** [50, 49, 47]

Abbreviations: PDC = Proportional-Derivative Control; KC = Kalman Control; FC = Fuzzy Control; PIC = Proportional-Integral Control; PIDC = Proportional-Integral-Derivative Control; LA = Lagrange Algorithm; ANN = Artificial Neural Network; MPC = Model Predictive Control; MA = Moving Average; QP = Quadratic Programming; MIMO = Multiple Input, Multiple Output; RL = Reinforcement Learning; DP = Dynamic Programming; ES = Exhaustive Search; ILP = Integer Linear Programming; MIP = Mixed Integer Programming; ACO = Ant Colony Optimization; DT = Decision Tree; NFM = Network Flow Model; FDS = Force-Directed Search; BS = Binary Search; LS = Local Search; GS = Grid Search; TS =Tabu Search; GA = Genetic Algorithm; PSO = Particle Swan Optimization; SMS-MOEA = S-metric Selection Multi-Objective Evolutionary Algorihtm; NSGA-II = Non-dominated Sorting Genetic Algorithm-II; MOACO = Multi-Objective Ant Colony Optimization; SAA = Sample Average Approximation.

From Tables 6 and 7, we can see that

(1) Most of the studies (78%) claim that they can work on any given objectives, and thus the QoS attributes and cost are the most popular objectives to be improved during the decision-making process of SSCAS.
(2) Hardware resources, particularly CPU and memory, are the most commonly considered dimension of control primitives to be tuned in SSCAS. However, there are few studies (9%) that consider the interplay between software and hardware control primitives.

Table 6. Decision-Making Approaches for SSCAS by Control Primitives and Objectives

Objective	*Bundles*	*Control Primitives*
Cost	Yes	**Number of VMs (3):** [30, 131, 123] **CPU and memory (2):** [132, 151] **CPU, memory, and number of VM (1):** [129, 105]
	No	**Number of VMs (1):** [116] **Configurations (1):** [59] **CPU (1):** [146]
Response time and cost	Yes	**Number of VMs (3):** [57, 89, 91] **Resources (1):** [148]
	No	**Number of VM (1):** [155] **CPU, memory, and bandwidth (1):** [83] **CPU, memory, and number of VM (1):** [75]
QoS attributes and cost	Yes	**Number of VMs (1):** [28] **Configurations and resources (1):** [94] **CPU and memory (1):** [69] **CPU, memory, and number of VMs (2):** [73, 29] **CPU, memory, and disk (1):** [73] **CPU, memory, and bandwidth (2):** [120, 61]
	No	**CPU (4):** [26, 104, 126, 24] **Number of VM (2):** [81, 64] **Resources (2):** [41, 119] **CPU and memory (6):** [74, 118, 149, 125, 100, 72] **Configurations and resources (4):** [50, 49, 47, 46] **CPU, memory and disk (1):** [121] **CPU, memory, and configurations (1):** [154] **CPU, storage, and bandwidth (1):** [65] **CPU, memory, and thread (2):** [140, 147] **CPU, bandwidth, storage, and number of VM (1):** [37] **CPU, thread, session, buffer, and memory (1):** [39].
QoS attributes	No	**Configurations (1):** [82] **Configurations and resources (1):** [152] **CPU, memory, and bandwidth (1):** [70] **Thread, CPU, and memory (1):** [43] **CPU and memory (1):** [99] **Resources (1):** [79] **Number of VMs (1):** [124]
Response time	No	**CPU (1):** [92] **Memory (1):** [68] **CPU and bandwidth (2):** [141, 127] **CPU and thread (1):** [103] **CPU and number of VMs (1):** [76] **CPU, memory, and number of VMs (1):** [106]
	Yes	**CPU, memory, and number of VMs (1):** [33]

Table 7. Decision-Making Approaches for SSCAS by Control Primitives and Objectives (continued)

Objective	*Bundles*	*Control Primitives*
CPU utilization	No	**CPU and number of VMs (1):** [107]
CPU utilization	Yes	**Number of VMs (1):** [63] **CPU and number of VMs (1):** [110]
Throughput	Yes	**Number of VMs (1):** [130]
General utilization	No	**Configurations (1):** [111] **Number of VMs (1):** [25] **Resources (1):** [80] **CPU (2):** [42, 84] **CPU and memory (1):** [136]
General utilization	Yes	**Number of VMs (1):** [90] **CPU, memory, number of VMs, and bandwidth (1):** [150]
QoS attributes and power	Yes	**CPU, memory, and number of VM (1):** [143]
Response time and utilization	Yes	**Number of VM (1):** [31] **Number of VM and VM type (1):** [32]
Response time, cost, and availability	No	**CPU and memory (1):** [21]
VM consumption	Yes	**Number of VM (2):** [35, 139]
SLA penalty	No	**CPU, memory, and number of VM (1):** [144]
Response time, throughput, CPU utilization, and cost	No	**CPU and memory (1):** [88]
Response time, throughput, CPU utilization	Yes	**Number of VMs and VM type (1):** [135]
SLA and power	No	**Resources (1):** [60]
SLA and power	No	**CPU (1):** [133]
Response time, throughput, and cost	No	**CPU and memory (1):** [138]
Benefits and overhead	No	**CPU and memory (1):** [20]
QoS attributes, utilization, and cost	No	**CPU (1):** [86]
QoS attributes, utilization, number of actions	No	**CPU, memory, and bandwidth (1):** [115]

(3) A considerable amount of studies (34%) has assumed bundles on the autoscaling decision, which will reduce the search space but might negatively constrain the quality of decision making.

Observations from Table 8 shows us that

(1) The majority of the studies (66%) has considered both vertical and horizontal scaling.
(2) Horizontal scaling receives much more attention than vertical scaling.

In the following, we specify some of the decision-making approaches for SSCAS in detail.

Table 8. Decision-Making Approaches for SSCAS by Scaling Actions

Scaling	*Decision-Making Approaches*
Vertical	**Rules (1):** [133] **Control theory (4):** [92, 82, 68, 99] **Search-based optimization (6):** [149, 41, 103, 88, 42, 111]
Horizontal	**Rules (6):** [84, 143, 130, 31, 63, 124] **Control theory (1):** [28] **Search-based optimization (13):** [57, 151, 81, 139, 155, 89, 91, 30, 25, 90, 131, 116, 135]
Both	**Rules (12):** [73, 62, 70, 65, 37, 83, 115, 43, 136, 123, 79, 106] **Control theory (14):** [26, 107, 29, 140, 118, 75, 146, 126, 154, 100, 24, 141, 127, 33] **Search-based optimization (35):** [147, 125, 132, 148, 120, 105, 74, 104, 94, 21, 152, 59, 119, 69, 121, 61, 64, 20, 138, 72, 86, 39, 80, 60, 129, 46, 114, 144, 50, 49, 47, 76, 110, 150, 32]

4.7.1 Rule-Based Control. Rule-based control is the most classic approach for making decision in SSCAS. Commonly, one or more conditions are manually specified and mapped to a decision, e.g., increase CPU and memory by x if the throughput is lower than y. Therefore, the possible tradeoff is often implicitly handled by the conditions and actions mapping. Specifically, Cloudline [73] allows programmable elasticity rules to drive autoscaling decisions. It is also possible to modify these rules at runtime as required by the users. Copil et al. [62] handle the decision-making process by specifying different condition-and-actions mapping for autoscaling in the cloud. In addition, the rules can be defined at different levels, e.g., PaaS and IaaS. Similarly, Ferretti et al. [70] set up mapping between QoS expectations and actions using XML-like notations.

4.7.2 Control-Theoretic Approach. Advanced control theory is another widely investigated approach for autoscaling decision making in SSCAS because of its low latency and dynamic nature. Studies in this category could be either *standard*, i.e., they rely solely on the classic control theory, or *extended*, where additional methods are considered.

Among the others, standard controllers (e.g., Proportional-Derivative control [28, 107, 29], Kalman control [92, 75], and Fuzzy control [26, 24, 140]) are commonly designed as a sole approach to make autoscaling decisions in the cloud. Specifically, ARUVE [28] and CRAMP [29] utilize a Proportional-Derivative (PD) controller, where the proportional and derivative factors are not sensitive to a concrete QoS model while supporting proactive autoscaling of cloud services and applications in a shared hosting environment. Anglano et al. [26] and Albano et al. [24] apply fuzzy control that is updated by fuzzy rules at runtime. The aim is to optimize QoS, cost, and energy by autoscaling hardware resources. Although the authors claim they can cope with any hardware resources, only CPU tuning is explored. They have also ignored the QoS interference.

We have also found that control-theoretic approaches can be sometime used with other algorithms to better facilitate the autoscaling decision making, forming an extended controller [146, 118, 82, 126, 154, 100]. Particularly, the gains in the controllers can be further tuned by optimization and/or machine-learning algorithms, and this is especially useful for Model Predictive Control (MPC). Among others, Zhu and Agrawal [154] utilize a Proportional-Integral-Derivative (PID) and reinforcement learning controller for decision making with respect to adapting software control primitives. Such a result is then tuned in conjunction with the hardware control primitives using exhaustive search. The QoS attributes and cost are formulated as a weighted-sum relation. The autoscaling decision-making process in APPLEware [100] has relied on Model Predictive Control

(MPC), with the aim to optimize a cost function that represents the local objectives and resource constraints at a point in time. The state of an application, together with the other autoscaling decisions from the neighboring VMs, are collectively considered in a quadratic programming solver.

4.7.3 Search-Based Optimization. A large amount of existing studies of SSCAS relies on search-based optimization, in which the decisions and tradeoffs are extensively reasoned in a finite, but possibly large, search space. Depending on the algorithms, search-based optimization for autoscaling decision making in the cloud can be either *explicit* or *implicit*—the former performs optimization as guided by explicit system models while this process is not required for the later.

- *4.7.3.1 Implicit search:* The implicit and search-based optimization approaches for autoscaling decision making do not use QoS models. Similarly to the control-theoretic approaches, the implicit search is also limited in reasoning about the possible tradeoffs. For example, the study from Xu et al. [125, 147] applies a model-free Reinforcement Learning (RL) approach for adapting thread, CPU, and memory for QoS and cost. The approach is, however, implicit, providing that there is neither explicit system models nor explicit optimization. The authors have considered QoS interference during autoscaling. Similarly, VScale [149] utilizes RL for making autoscaling decisions, which are then achieved by vertical scaling. The RL is realized by using parallel learning; that is, the authors intend to speed up agent's learning process of approximated model by learning in parallel, without visiting every state-action pair in a given environment.

  The approaches that rely on demand prediction (e.g., the Autoflex [25] and PRESS [80, 129, 42, 60, 90]) are also regarded as implicit search. This is because the autoscaling decision is directly predicted by the demand models, without the needs of reasoning and optimization.
- *4.7.3.2 Explicit search:* In search-based optimization category, the explicit approaches for autoscaling decision making rely on the explicit QoS models to evaluate and guide the search process. Depending on the different formulations of the decision-making problem for autoscaling in the cloud, explicit search can reason about the effects of decisions and the possible tradeoffs in detail. According to our survey, we found the three most commonly used formulations, which are single-objective optimization, weighted-sum optimization, and Pareto-based optimization.

  We discovered that it is not uncommon to optimize only a single objective (e.g., cost or profit) for SSCAS, providing that the requirements of the other objectives are satisfied (i.e., they are often regarded as constraints). For example, Kingfisher [132] and Sedaghat et al. [129] use Integer Linear Programming (ILP) to optimize the cost for scaling the CPU and memory for VMs of an application while regarding the demand for satisfying QoS as constraint.

  To apply explicit search-based optimization for SSCAS, the most widely solution for handling the multi-objectivity is to aggregate all related objectives into a weighted (usually weighted-sum) formulation, which converts the decision-making process into a single-objective optimization problem. The search-based algorithms include the following: exhaustive search [57, 41, 74, 89], the auxiliary network flow model [104], force-directed search [81], and binary search [91]. For example, the FoSII project [37, 114] regards autoscaling decision making as a case-based reasoning process, where the decision is made by looking for similar historical cases using exhaustive search. The solution of the most similar case is reused to solve the current one.

  We noted that some studies have relied on more advanced and nonlinear search algorithms, ranging from relatively simple ones, such as dynamic programming [120] and local-search strategy [21], to more complex forms, such as grid search [144], decision

Table 9. Infrastructure, Benchmarks, and Workload Traces for Experimental Evaluation on SSCAS

Experiment Setup		*Software and Tools*
Infrastructure	Simulator	CloudSim [40], CDOSim [71], CloudAnalyst [142], DCSim [93]
	Private cloud	Xen [19], VMWare ESXi [15], KVM [7], Docker [4], OpenStack [9], Eucalyptus [5]
	Public cloud	Amazon EC2 [1], RackSpace [10], Azure [8], Google Compute Engine [6]
Benchmarks		RUBiS [11], RUBBoS [12], TPC-W [13], WikiBench [16]
Workload traces		Synthetic trace, FIFA98 [18], Wikipedia [17], ClarkNet [3]

tree search [69, 111], and quadratic programming [121]. For example, CLOUDFARM [120] addresses the decision making based on a weighted-sum utility function of all cloud-based application and services. The decision-making process is formulated as a knapsack problem, which can be resolved by dynamic programming.

We have also found that metaheuristic algorithms are popular for autoscaling decision making in SSCAS, because they can often efficiently address NP-hard problems with approximated results under no assumptions of the problem. The most common algorithms include the following: Tabu Search [155], Genetic Algorithm (GA) [152, 119, 20], and Particle Swarm optimization (PSO) [152]. As an example, Zhu et al. [155] formulate the autoscaling decision making as optimize a weighted-sum formulation of response time and cost. To optimize the objectives, the authors apply a hybrid Tabu Search, which relied on iterative gradient descent.

Finally, the Pareto relation can explicitly handle multi-objectivity for autoscaling in the cloud without the need to specify weights on the objectives [64, 103, 138, 72, 49]. For example, in E^3-R [138], the decision-making problem is formulated using the Pareto relation, where it is resolved by using Non-dominated Sorting Genetic Algorithm-II (NSGA-II). Further, the approach applies an objective reduction technique with an aim to remove the objectives, which are not significantly conflicted with the others, from the decision-making process. Chen and Bahsoon [49] exploit Multi-Objective Ant Colony Optimization (MOACO) for tradeoff decision making when autoscaling cloud-based services. The authors consider the tradeoff between naturally conflicting objectives and between competing services (i.e., QoS interference). Further, a compromise-dominance mechanism is proposed to find a well-compromised tradeoff decision.

4.8 Experimental Evaluation on SSCAS

Another important step in SSCAS research is to quantitatively evaluate the proposed solution, which is often achieved via experimental analysis. To this end, setting the experiments are of high importance to researchers in this field. Table 9 summarizes the common infrastructure, benchmarks, and workload trace from the considered studies. We can see that studies have chosen to use a simulator for controlled experiments and ease of complexity. However, simulators may not fully capture the realistic environment. As a result, custom private cloud and public cloud are also exploited for both controlled and open experiments. Notably, custom private cloud can be much more flexible on choosing the underlying software, e.g., one may utilize the hypervisor or container directly or choose to use higher level software such as OpenStack [9]. It is worth noting that custom private and public cloud can share similar underlying software and tools, but they

may require expertise on different levels of abstraction. For example, one may choose to deploy SSCAS on a custom private cloud that makes use of Xen [19]—the same hypervisor that underpins Amazon EC2 [1], which contains additional high-level interfaces and restrictions. Benchmarks are not required for a simulator, but it is crucial for both private and public cloud infrastructure. A wide range of benchmarks have been exploited to evaluate SSCAS, from simple web hosting to complex multi-tier software. Finally, the workload traces can be either synthetic, in which a fixed pattern is generated by the workload generator (e.g., JMeter [2]), or real, where recorded traces form different real domains are used to stress the benchmark and SSCAS.

4.9 Implementation of Scaling

The actual implementation of scaling depends on the underlying scenarios, e.g., the type of hypervisor/container, the cloud-based applications, and the actual cloud control primitives among the others. Existing virtulization techniques have provided readily available commands and tools to support autoscaling at runtime. For example, if the underlying hypervisor was Xen [19], then resources such as CPU and memory of a VM, as well as create/destroy VMs, can be scaled dynamically using the *xm* command. Regarding the actual scaling methods, vertical scaling will have trivial effects on the states of the cloud-based applications, and thus they can be directly applied using the command support by hypervisor/container. For horizontal scaling, making new replicas or removing old one needs consistency guarantee on stateful applications/services, which can be ensured by the underlying hypervisor through various readily available protocols. For example, in Xen, horizontal scaling can be achieved via *primary-backup replication* or *asynchronous checkpointing*, *and so on.*

5 REFLECTIONS AND OPEN CHALLENGES FOR SSCAS RESEARCH

In this section, we reflect on the finding of our survey and taxonomy; state the open challenges as well as discuss industrial situation and pricing strategy for SSCAS.

5.1 Discussion and Comparison on Existing SSCAS Research

We now discuss the most noticeable observations by reviewing the existing SSCAS research. We carefully position our discussions in light of the different logical aspects of SSCAS.

5.1.1 The Levels of Self-Awareness and Self-Adaptivity in Cloud Autoscaling Systems. *Stimulus-awareness* has been considered in all the 109 studies as it is the most fundamental levels in self-awareness principles, because it is the basic requirement for a software system to be able to adapt. *Time-* (52%) and *goal-awareness* (52%) receives same attention due to the fact that the objective models often contain historical information and they can be used to reason about goals. In contrast, handling *interaction-* (13%) and *meta-self-awareness* (7%) are less popular, as the former requires us to handle QoS interference while the latter often comes with extra complexity.

While *self-optimizing* and *self-configuring* have been the major themes for SSCAS, we found very few studies that considered *self-healing* (5%), and only one work targets *self-protecting*. This is obvious, as the fundamental idea of autoscaling is not for security-related purposes but for the performance related quality, which is often much more appealing for cloud consumers.

Also, 67% of the studies has ignored the importance of specifying the required knowledge at the architecture level, entailing the risk of limited awareness [34].

5.1.2 Architectural Pattern. The generic feedback (82%), particularly the single and close loop, has been the predominant architectural pattern for SSCAS. MAPE (15%) is ranked second, and OAD (3%) is the much less popular one, as OAD often assumes the involvement of human decisions makers, which is difficult in the case of SSCAS.

The reason could be due to the fact that the feedback loops are flexible and simple to be realized, providing the basic components to achieve self-adaptivity. However, such a design can limit the consideration of the required knowledge for the autoscaling system to perform adaptations, or the consideration is rather simple and coarse grained. In contrast, such an issue has been relaxed by OAD and MAPE, as they are more restricted by predefining components to capture different aspects of a SSCAS. We see that MAPE is clearly more popular than OAD, because the former can be good for separation of concepts (e.g., *Analyze*, *Plan*, and *Knowledge*) and for expressing the sequential interactions between those concepts while the latter fails to capture runtime aspects of the SSCAS, as it is mainly designed for decoupling loops of different human activities. Nevertheless, these architectural patterns lack fine-grained representation of the required knowledge. Thus, it is not immediately intuitive what level(s) of knowledge is required by each logical aspect of a SSCAS.

5.1.3 QoS Modeling. Both the analytical model and the machine-learning model, included in 43% and 38% of the studies respectively, are widely exploited in QoS modeling for SSCAS while the simulation (10%) and hybrid (9%) approaches are clearly less popular. This could be because analytical model is good for runtime efficiency, simplicity, and interoperability, and they could be very effective if all of their assumptions are satisfied. However, analytical approaches generally require in-depth knowledge about the likely behaviors of the system being modeled, i.e., some knowledge about the system's internal structure or environmental conditions. Such an issue is resolved by using machine-learning models that are often assumptions free, and, more importantly, they are able to continually evolve themselves at runtime to cope with dynamics and uncertainty. Nevertheless, depending on the learning algorithm, the resulting overhead can be high (e.g., the nonlinear ones), and the accuracy is sensitive to the situation (e.g., fluctuation of the data trend).

In contrast, simulation exhibits static nature and it is restricted by a wide set of assumptions, including, e.g., the distribution of workload and the effects of QoS interference, and so on, needs complex human intervention and assumptions. However, it is believed that the simulations model could be the most accurate way to model QoS when the all assumptions are satisfied [71]. Hybrid models, as in six of the studies surveyed, could potentially combine the strengths from different models.

We noted that only 13% of the studies intend to address QoS interferences when modeling the QoS in SSCAS. This might be because considering QoS interference will significantly increase the dimensionality in the model, which in turn, renders the problem much more complex. Such a complexity makes human analysis very difficult if not impossible. As a result, for those studies that do consider QoS interference, machine-learning algorithms are often exploited.

There are plenty of studies (61%) that consider the dynamic (or semi-dynamic) structure of a QoS model (i.e., those that denoted as both *dynamic* and *semi* in Table 2); however, the dynamic related to the input features have been rarely researched simultaneously, i.e., only 7% of the studies (for those that denoted as *dynamic* in Table 2). Indeed, changing the inputs of a model could be useful only when the dimensionality of a model is high; that is, changing the input features might not cause a significant difference if the considered total number of inputs is less than 5. The majority of the *dynamic* (or *semi-dynamic*) modeling are machine-learning based, leaving the *static* ones largely analytical or simulation based. This is obvious, as the nature of those modeling approaches determines the extent to which they can be changed when they are built.

A considerable amount of studies (65%) have claimed that their QoS models can work on any given inputs and/or output, as shown in Table 3. This happens mostly for machine-learning and simulation approaches. However, during experimental analysis, the highest number of QoSs being modeled and the number of inputs were both four [52, 48]. The most commonly considered output is response time, and the inputs are hardware resources, particularly CPU, memory, and number of VM. This complies with the current trend in the cloud computing market.

5.1.4 Granularity of Control. In SSCAS research, the service/application level of granularity is the most popular one, which yields 45% of the studies. This is because focusing on the finest granularity of control can achieve the maximum level of scalability, which particularly fits the cloud. However, fine granularity of control is achieved in the expenses of the globally optimal quality of the cloud, since no interactions between service and application are considered. In contrast, focusing on cloud level is another extreme that trades scalability for global optimality. Considering multiple levels could be a solution to reach a better tradeoff as discussed in a small amount of studies (8%), but how and when to select the levels to consider imposes additional challenges.

Notably, most of the studies (72%) have relied on the control of each application in cloud regardless to the actual granularity of control. The reason being could be due to the fact that a cloud-based application (or a collection of services) is the most crucial unit for consumers to experience the benefit of the cloud, which is a common interest for both cloud consumers and providers.

5.1.5 Decision Making. Generally, rule-based control is a highly intuitive approach for autoscaling decision making, and it also has negligible overhead. However, the static nature of the rules requires us to assume all the possible conditions and the effects of those decisions that are mapped to the conditions, which is highly dependent on the assumptions. To resolve such an issue, a control-theoretic approach appears to be an effective solution, as it is also efficient while requiring very few assumptions. However, the major drawback of control-theoretic approaches is that they require us to make many actuations on the physical system to collect the "errors" to stabilize itself. This means that amateur decisions are very likely to be made. In addition, both approaches lack multi-objectivity and tradeoffs; they often fail to cope with the problem where there is a large number of autoscaling decisions, which is common for the cloud. In contrast, search-based optimization, especially explicit search, makes loose assumptions about the number of autoscaling decisions and is able to find optimality (or near-optimality) under highly dynamic and uncertain environments. Therefore, as we have shown, search-based optimization, either implicit or explicit, is the most popular approach for making decisions in SSCAS.

We noted that there is a considerable amount of studies (63%) that do not attempt to explicitly consider tradeoff during decision making of SSCAS, as they assumed single objective or rely completely on human preferences. The reason might be attributed to the fact that such a formalization is simple and straightforward, which can work well when there is a strong preference on an objective. The rest of the studies handle multi-objectivity via either weighted sum or the Pareto relation.

QoS interference is again absent in many studies (84%) due to the fact that considering it will unavoidably increase the dimensionality (i.e., objectives) during decision making, leading to a more complex problem. This would make the problem unsolvable by many existing approaches.

We found that most of the studies (78%) have claimed that their decision-making approach could handle any given objectives, and thus they have considered arbitrary QoS attributes and cost as the objectives in SSCAS, since these are the most critical indicators for cloud-based services and applications. The highest number of objectives that were considered during the experiments are 5 [49], though. CPU, memory, and number of VM are the most popular control primitives in decision making, because they are the most straightforward dimension to be scaled. However, only as little as 9% of the studies consider the interplay between software and hardware control primitives. To apply solution that cannot handle a large search space of the decision making, one often reduces the search space by introducing fixed bundles, which is an assumption made by a considerable amount of studies, i.e., 34%. However, such a reduction has the risk that some good solution can be ruled out during the process. Finally, while both vertical and horizontal scaling have been considered in the majority of the studies (66%), focusing solely on horizontal scaling is more popular than the vertical one, as the former is more widely supported by major cloud vendors.

5.2 Open Problems and Challenges for SSCAS Research

Drawing on the survey and taxonomy, in the following, we specify the open problems and challenges for future SSCAS research and make suggestions for potential research directions where appropriate.

- *Explicit Knowledge Representations Are Required in SSCAS Architecture:* As we can see from Section 4.3, only 33% of the studies intend to discuss the required knowledge at the architecture level. This means that, in the remaining 67% of the work, it is more difficult to capture more complex and advanced levels of knowledge, as evident by the fact that most work does not go beyond the basic *stimulus-awareness*. Indeed, studies in References [34, 54, 102, 101, 53] have found that, for self-aware and self-adaptive software systems in general, the absence of explicit consideration for the fine-grained representation of the knowledge in the architecture can result in, e.g., improper inclusion of unnecessary knowledge and/or missing important knowledge that can improve adaptation quality when developing autoscaling systems; 67% of studies that do not discuss knowledge at the architecture level implies that such an issue is often overlooked and remains unresolved in the SSCAS context, urging the need for further investigation.

  The challenge here lies in how we can systematically distinguish different levels of knowledge and how they can be built into SSCAS in a principal way. We argue that the required levels of knowledge and their representations can be declared with the formal principle of self-awareness. In particular, a potential way is to follow the handbook [54] for mapping different levels of knowledge into a concrete SSCAS architecture.
- *Multiple Loops Can Create More Benefit for SSCAS Architecture:* From Section 4.4, we noted that the majority of the studies has considered single loop, which could cause the problem of high coupling in the design of SSCAS. The multiple loops, however, help to achieve better separation between different aspects of SSCAS, leading to fine-grained and localized adaptation. The challenge here is how many and at what levels of abstraction one should place the loops within the SSCAS architecture. We suggest that designing multiple looped SSCAS architectures with respect to what levels of knowledge the system required could be a nice solution [47].
- *QoS Interference Should Be Explicitly Handled in QoS Modeling and Decision-Making Processes:* Our survey results (see Sections 4.5 and 4.7) indicate that only less than 16% of the studies took QoS interference into account. Missing QoS interference in the model and decision making could lead to incorrect or misled autoscaling decisions, as the cloud-based services would be unavoidably affected by the dynamic behaviors of its neighbors. However, incorporating QoS interference rises the challenge of dimensionality, which makes the modeling and decision making much more complex. Therefore, this challenge calls for a novel approach to reduce the dimensionality, or mitigate its negative effects, during the modeling and decision making in SSCAS [48, 49].
- *The Interplay between Software Configuration and Hardware Resources Is Important:* Most existing studies of SSCAS focus on hardware resources as an IaaS level only. However, as shown in References [39, 152, 111, 48], various software configurations at the PaaS level could interplay with each other and the hardware resources, which, in turn, affects the QoS of cloud-based services. The challenge is how to create a holistic approach that combines both the PaaS and IaaS levels in SSCAS.
- *Dynamic Feature Selection Is Required for QoS Modeling in SSCAS:* Section 4.5 indicates that the majority of the existing studies have ignored primitive selection in the QoS modeling or have been relying on a manual approach, because the assumed dimensions of inputs

are rather limited. However, when both QoS interference and software configurations are considered, selecting the most significant features in the model becomes a crucial task, since there could be an explosion of the primitives space [48, 145]. The challenge here lies in how to evaluate the effectiveness of feature combination on model accuracy while generating reasonable overhead. Since the type of features can be arbitrary, it is important to design generic and efficient feature selection approach for SSCAS.

- *More Flexible Granularity of Control in SSCAS Is Needed:* Single, static, and fixed granularity of control is predominately exploited in existing SSCAS research. To better handle dynamic and uncertainty, it could be more beneficial to introduce multiple granularity and/or dynamically adjust the granularity of control at runtime [144, 46], as the granularity of control implies a tradeoff between the global optimality of SSCAS and the imposed overhead. The challenge is how to explicitly capture the objective-dependency when designing granularity of control.
- *The Assumptions on the Bundles of Resources Needs to Be Relaxed:* From Section 4.7, we noted that while most of the studies have not constrained the possible autoscaling decisions with respect to the fixed bundles, there are still certain studies that heavily rely on the fixed types of bundles, e.g., a search space of 57 VM instance types on Amazon EC2. However, renting bundles cannot and does not reflect the interests of consumers and the actual demand of their cloud-based services. We argue that future cloud autoscaling would inevitably need to take an arbitrary combination of software configurations and resources into account, as has already been supported in Google Compute Engine [6]. As a result, autoscaling decision making imposes a challenging problem that is faced with an explosion of decision space (e.g., millions of alternatives), calling for a novel and efficient approach to achieve optimal or near-optimal quality.
- *The Tradeoff between Conflicting Objectives Should Be Explicitly Handled:* As shown in Section 4.7, the approaches have mostly ignored tradeoff. There is also a certain amount of explicit search-based optimization studies that has assumed only single objective. However, this can restrict the applicability of SSCAS as the decision making would fail in identifying good tradeoff points or strongly bias to the single objective. Alternatively, there is also a considerably large amount of studies that exploit weighted sum objective aggregation, which embeds the tradeoff in a single representation. However, it is well known that the relative weights are difficult to tailor, and a single aggregation could restrict the search, causing limitations when searching for good decisions spread over the search space. Further, achieving balanced tradeoff has only being explored in very limited studies, e.g., the work from Chen and Bahsoon [49]. The challenge here is to search for decisions that contain good convergence and diversity and eventually select the one that has the most balanced tradeoff for scaling. We advocate that stochastic optimization approaches, particularly nature inspired algorithms, can be promising in addressing such a challenge.
- *More Real World Case Studies of SSCAS Are Needed:* We found that real-world cases and scenarios of SSCAS, especially those with large scale and practical application, are absent in many studies. Indeed, those studies impose many challenges beyond the perspective of research, but they can be the only way to fully verify the potentials, effectiveness, and impacts of SSCAS.

5.3 Current Industrial Situation of SSCAS

Industrial cloud providers (e.g., Amazon [1] and RightScale [14]) have been relying on model-free, simple rule, and policy-based autoscaling approaches for decades. These approaches dismiss the difficult problem of how to specify rules to cloud consumers, which may work well in the beginning

when the demand and complexity of cloud-based applications are simple and straightforward. However, recently the level of complexity (e.g., in terms of the number of cloud control primitives) of the cloud is changing to a state that makes human analysis very difficult, especially under conflicting objectives and a large number of alternative autoscaling decisions [64, 103, 138, 49]. Specifically, as discussed in Section 5.1.5, those approaches suffer two significant pitfalls as follows: (i) They require understanding of the application and domain knowledge to determine the mapping between conditions and actions, which can significantly affect the quality of scaling [23, 152], and (ii) they cannot adapt to dynamically changing workload or state of the applications [119, 49].

As a result, engineering advanced SSCAS is an inevitable trend in this area; the reason current big cloud providers have not yet widely implemented them could be due to the fact that SSCAS itself is not mature to the state where it can be reliably adopted. However, as our survey reports, researchers and practitioners have been working on overcoming these challenges for almost a decade. There have been some attempts to apply SSCAS commercially; for example, Microsoft Azure [8] has recently benefited from Aneka [137], a research effort supporting high-level framework that contains a more advanced and complicated SSCAS that relies on search-based optimization as part of its subsystems. Aneka's work is evidence of how pending industrial challenges had informed research; the results are now incorporated into Azure. Nevertheless, we envision more progress on enhanced, scalable, and cost-effective effective solutions for both the cloud providers and consumers.

5.4 Discussion on the Pricing for Cloud Autoscaling

Indeed, more advanced autoscaling approaches in SSCAS (e.g., machine-learning and search-based optimization) may impose additional computational resources. Furthermore, advances in autoscaling cannot be done in isolation of pricing (dynamic metering and pricing in particular), as both the cost and revenue are acknowledged among the drivers for the industrial need of more advanced solutions. However, upfront investment in additional recourse can be arguably paid off in situations where scale and dynamic demand is effectively enabled. This can be observed through greater payoff through better utilization and SLA guarantee, which in turn improves the overall reputation and thus attracts more consumers. As analyzed in numerous existing works [109, 122, 72, 48, 49], the additional resources spent are actually marginal compared with the savings obtained through more accurate, effective autoscaling. In case the cloud provider wishes to charge the consumers for the services provided by SSCAS, there could be two ways to achieve this. (i) The first method is to charge the computation utilized by the SSCAS through an existing pricing schema, e.g., the *reserved* or *spot instance* from Amazon EC2. Here, the SSCAS is an optional service that would be priced as a normal instance for the consumers' application/services. (ii) The second method is to charge SSCAS in combination with the normal price per time unit in an existing pricing schema, e.g., instead of charging \$1/hour of an instance, it can be priced as \$1.3/hour, where the extra \$0.3/hour is for the SSCAS. Here, the SSCAS is a default and mandatory service for cloud consumers.

6 CONCLUSION

In this article, we survey the state-of-the-art research on SSCAS and provide a taxonomy based on our findings. Specifically, we review the literature with respect to the research questions presented in Section 4.1. According to our survey, the key findings are as follows:

— *Stimulus-*, *time-*, and *goal-awareness* are the most widely considered levels of knowledge in SSCAS. *Self-configuring* and *Self-optimizing* are the most popular self-adaptivity notions in SSCAS.

—Feedback loop is the most commonly exploited architectural pattern for engineering SSCAS.
—Analytical models and machine-learning-based models are prominent for QoS modeling in SSCAS.
—Controlling at the level of service/application is the mostly applied granularity.
—Search-based optimization is the most common approach for making autoscaling decisions.

Apart from those observations, we also gain many insights on the open problems and challenges for future SSCAS research. The most noticeable ones are as follows:

—Explicit knowledge representations are required in SSCAS architecture.
—Multiple loops can create non-trivial benefits for SSCAS architecture.
—QoS Interference should be explicitly handled in the QoS modeling and decision-making process.
—The interplay between software configurations and hardware resources is non-trivial.
—Dynamic feature selection is required for QoS modeling in SSCAS.
—More flexible granularity of control in SSCAS is needed.
—The assumptions on the bundles of resources need to be relaxed.
—The tradeoff between conflicting objectives should be explicitly handled.
—More real-world cases and scenarios of SSCAS are needed.

We hope that our survey and taxonomy will motivate further research for a more intelligent cloud autoscaling system and its interactions with the other problems in the cloud computing paradigm.

REFERENCES

[1] Amazon. Amazon Elastic Compute Cloud. Retrieved March 24, 2018 from http://aws.amazon.com/ec2/.
[2] Apache. Apache JMeter. Retrieved March 24, 2018 from http://jmeter.apache.org/.
[3] Lawrence Berkeley National Laboratory. ClarkNet HTTP Trace. Retrieved March 24, 2018 from http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html.
[4] Docker. Docker: The Cloud Container. Retrieved March 24, 2018 from https://www.docker.com/.
[5] Eucalyptus Systems. Eucalyptus Cloud. Retrieved March 24, 2018 from http://www.eucalyptus.com/.
[6] Google. Google Compute Engine. Retrieved March 24, 2018 from http://cloud.google.com/products/compute-engine.html/.
[7] Red Hat. Kernel Based Virtual Machine. Retrieved March 24, 2018 from http://www.linux-kvm.org/.
[8] Microsoft. Microsoft Windows Azure. Retrieved March 24, 2018 from https://www.windowsazure.com/en-us/.
[9] RackSpace. OpenStack Framework. Retrieved March 24, 2018 from https://www.openstack.org/.
[10] RackSpace. RackSpace Cloud. Retrieved March 24, 2018 from https://www.rackspace.com/.
[11] Rice University. Rice University Bidding Systems. Retrieved March 24, 2018 from http://rubis.ow2.org/.
[12] Rice University. RUBBoS: Bulletin Board Benchmark. Retrieved March 24, 2018 from http://jmob.ow2.org/rubbos.html/.
[13] TPC. TPC-W. Retrieved March 24, 2018 from http://www.tpc.org/tpcw/default.asp.
[14] RightScale. Understanding the Voting Process. Retrieved March 24, 2018 from https://support.rightscale.com/12-Guides/RightScale_101/System_Architecture/RightScale_Alert_System/Alerts_based_on_Voting_Tags/Understanding_the_Voting_Process/.
[15] VMware. VMware vSphere ESX and ESXi. Retrieved March 24, 2018 from https://www.vmware.com/products/esxi-and-esx.html.
[16] Guillaume Pierre. WikiBench: A Web hosting benchmark. Retrieved March 24, 2018 from http://www.wikibench.eu.
[17] Guillaume Pierre. Wikipedia access traces. Retrieved March 24, 2018 from http://www.wikibench.eu/?pageid=60.
[18] Lawrence Berkeley National Laboratory. World Cup 98 Trace. Retrieved March 24, 2018 from http://ita.ee.lbl.gov/html/contrib/WorldCup.html.
[19] Linux Foundation. Xen: a virtual machine monitor. Retrieved March 24, 2018 from http://xen.xensource.com/.
[20] Omar Abdul-Rahman, Masaharu Munetomo, and Kiyoshi Akama. 2012. Toward a genetic algorithm based flexible approach for the management of virtualized application environments in cloud platforms. In *Proceedings of the 2012*

21st International Conference on Computer Communications and Networks (ICCCN'12). 1–9. DOI: http://dx.doi.org/10.1109/ICCCN.2012.6289218

[21] Bernardetta Addis, Danilo Ardagna, Barbara Panicucci, and Li Zhang. 2010. Autonomic management of cloud service centers with availability guarantees. In *2010 IEEE 3rd International Conference on Cloud Computing*. 220–227. DOI: http://dx.doi.org/10.1109/CLOUD.2010.19

[22] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. 2018. Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing* 11, 2 (2018), 430–447.

[23] Fahd Al-Haidari, M. Sqalli, and Khaled Salah. 2013. Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources. In *IEEE 5th International Conference on Cloud Computing Technology and Science*. 256–261.

[24] Luca Albano, Cosimo Anglano, Massimo Canonico, and Marco Guazzone. 2013. Fuzzy-Q&E: Achieving QoS guarantees and energy savings for cloud applications with fuzzy control. In *2013 Third International Conference on Cloud and Green Computing (CGC)*. 159–166. DOI: http://dx.doi.org/10.1109/CGC.2013.31

[25] Fabio Jorge Almeida Morais, Francisco Vilar Brasileiro, Raquel Vigolvino Lopes, Ricardo Araújo Santos, Wade Satterfield, and Leandro Rosa. 2013. Autoflex: Service agnostic auto-scaling framework for iaas deployment models. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 42–49. DOI: http://dx.doi.org/10.1109/CCGrid.2013.74

[26] Cosimo Anglano, Massimo Canonico, and Marco Guazzone. 2015. FC2Q: Exploiting fuzzy control in server consolidation for cloud applications with SLA constraints. *Concurr. Comput.: Pract. Exper.* 27, 17 (2015), 4491–4514.

[27] Danilo Ardagna, Giuliano Casale, Michele Ciavotta, Juan F. Pérez, and Weikun Wang. 2014. Quality-of-service in cloud computing: Modeling techniques and their applications. *J. Internet Serv. Appl.* 5, 1 (2014), 11.

[28] Adnan Ashraf, Benjamin Byholm, Joonas Lehtinen, and Ivan Porres. 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 (SEAA)*. 431–438. DOI: http://dx.doi.org/10.1109/SEAA.2012.13

[29] Adnan Ashraf, Benjamin Byholm, and Ivan Porres. 2012. CRAMP: Cost-efficient resource allocation for multiple web applications with proactive scaling. In *2012 IEEE 4th International Conference on Cloud Computing Technology and Science*. 581–586. DOI: http://dx.doi.org/10.1109/CloudCom.2012.6427605

[30] Adnan Ashraf and Ivan Porres. 2014. Using ant colony system to consolidate multiple web applications in a cloud environment. In *2014 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 482–489. DOI: http://dx.doi.org/10.1109/PDP.2014.101

[31] Mohammad Sadegh Aslanpour, Mostafa Ghobaei-Arani, and Adel Nadjaran Toosi. 2017. Auto-scaling web applications in clouds: A cost-aware approach. *J. Netw. Comput. Appl.* 95 (2017), 26–41.

[32] Uchechukwu Awada and Adam Barker. 2017. Improving resource efficiency of container-instance clusters on clouds. In *IEEE 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 929–934.

[33] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. 2016. A discrete-time feedback controller for containerized cloud applications. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 217–228.

[34] Tobias Becker, Andreas Agne, Peter R. Lewis, Rami Bahsoon, Funmilade Faniyi, Lukas Esterle, Ariane Keller, Arjun Chandra, Alexander R. Jensenius, and Stephan C. Stilkerich. 2012. EPiCS: Engineering proprioception in computing systems. In *2012 IEEE 15th International Conference on Computational Science and Engineering (CSE)*. 353–360. DOI: http://dx.doi.org/10.1109/ICCSE.2012.56

[35] Jing Bi, Zhiliang Zhu, Ruixiong Tian, and Qingbo Wang. 2010. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*. 370–377. DOI: http://dx.doi.org/10.1109/CLOUD.2010.53

[36] Peter Bodík, Rean Griffith, Charles Sutton, Armando Fox, Michael Jordan, and David Patterson. 2009. Statistical machine learning makes automatic control practical for internet datacenters. In *2009 Conference on Hot Topics in Cloud Computing*. USENIX Association, Berkeley, CA, Article 12. http://dl.acm.org/citation.cfm?id=1855533.1855545

[37] Ivona Brandic, Vincent C. Emeakaroha, Michael Maurer, Schahram Dustdar, Sandor Acs, Attila Kertesz, and Gabor Kecskemeti. 2010. LAYSI: A layered approach for SLA-violation propagation in self-manageable cloud infrastructures. In *IEEE 34th Annual Computer Software and Applications Conference Workshops*. 365–370. DOI: http://dx.doi.org/10.1109/COMPSACW.2010.70

[38] Yuriy Brun, Giovanna Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. 2009. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems*. Springer-Verlag, Berlin. 48–70. DOI: http://dx.doi.org/10.1007/978-3-642-02161-93

[39] Xiangping Bu, Jia Rao, and Cheng zhong Xu. 2013. Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. *IEEE Transactions on Parallel and Distributed Systems* 24, 4 (April 2013), 681–690. DOI: http://dx.doi.org/10.1109/TPDS.2012.174

[40] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Csar A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.* 41, 1 (Jan. 2011), 23–50. DOI: http://dx.doi.org/10.1002/spe.995

[41] Radu Calinescu, Lars Grunske, Marta Kwiatkowska, Raffaela Mirandola, and Giordano Tamburrelli. 2011. Dynamic QoS management and optimization in service-based systems. *IEEE Transactions on Software Engineering* 37, 3 (May 2011), 387–409. DOI: http://dx.doi.org/10.1109/TSE.2010.92

[42] Eddy Caron, Frederic Desprez, and Adrian Muresan. 2010. Forecasting for grid and cloud computing on-demand resources based on pattern matching. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*. 456–463. DOI: http://dx.doi.org/10.1109/CloudCom.2010.65

[43] Antonin Chazalet, Frédéric Dang Tran, Marina Deslaugiers, Alexandre Lefebvre, François Exertier, and Julien Legrand. 2011. Adding self-scaling capability to the cloud to meet service level agreements. *International Journal on Advances in Intelligent Systems*, 4, 3 (2011).

[44] Tao Chen and Rami Bahsoon. 2011. Scalable service oriented replication in the cloud. In *2011 IEEE International Conference on Cloud Computing (CLOUD)*. 766–767. DOI: http://dx.doi.org/10.1109/CLOUD.2011.114

[45] Tao Chen and Rami Bahsoon. 2013. Self-adaptive and sensitivity-aware QoS modeling for the cloud. In *8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 43–52. http://dl.acm.org/citation.cfm?id=2663546.2663556

[46] Tao Chen and Rami Bahsoon. 2014. Symbiotic and sensitivity-aware architecture for globally-optimal benefit in self-adaptive cloud. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 85–94. DOI: http://dx.doi.org/10.1145/2593929.2593931

[47] Tao Chen and Rami Bahsoon. 2015. Towards A smarter cloud: Self-aware autoscaling of cloud configurations and resources. *IEEE Comput.* 48, 9 (Sept 2015), 93–96.

[48] Tao Chen and Rami Bahsoon. 2017. Self-adaptive and online qos modeling for cloud-based software services. *IEEE Transactions on Software Engineering* 43, 5 (2017), 453–475.

[49] Tao Chen and Rami Bahsoon. 2017. Self-adaptive tradeoff decision making for autoscaling cloud-based services. *IEEE Transactions on Services Computing* 10, 4 (2017), 618–632.

[50] Tao Chen, Rami Bahsoon, and Georgios Theodoropoulos. 2013. Dynamic QoS optimization architecture for cloud-based DDDAS. *Procedia Computer Science* 18 (2013), 1881–1890. DOI: http://dx.doi.org/10.1016/j.procs.2013.05.357. 2013 International Conference on Computational Science.

[51] Tao Chen, Rami Bahsoon, Shuo Wang, and Xin Yao. 2018. To adapt or not to adapt? technical debt and learning driven self-adaptation for managing runtime performance. In *ACM/SPEC International Conference on Performance Engineering*.

[52] Tao Chen, Rami Bahsoon, and Xin Yao. 2014. Online QoS modeling in the cloud: A hybrid and adaptive multi-learners approach. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. 327–336. DOI: http://dx.doi.org/10.1109/UCC.2014.42

[53] Tao Chen, Funmilade Faniyi, and Rami Bahsoon. 2016. *Design Patterns and Primitives: Introduction of Components and Patterns for SACS*. Springer International Publishing, Cham, 53–78. DOI: http://dx.doi.org/10.1007/978-3-319-39675-05

[54] Tao Chen, Funmilade Faniyi, Rami Bahsoon, Peter R. Lewis, Xin Yao, Leandro L. Minku, and Lukas Esterle. 2014. The handbook of engineering self-aware and self-expressive systems. *Arxiv Preprint Arxiv:1409.1793* (2014).

[55] Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature guided and knee driven multi-objective optimization for self-adaptive software. (submitted).

[56] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. 2009. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee (Eds.). Lecture Notes in Computer Science, Vol. 5525. Springer Berlin Heidelberg, 1–26. DOI: http://dx.doi.org/10.1007/978-3-642-02161-91

[57] Ruiqing Chi, Zhuzhong Qian, and Sanglu Lu. 2012. A game theoretical method for auto-scaling of multi-tiers web applications in cloud. In *Fourth Asia-Pacific Symposium on Internetware*. Article 3, 10 pages. DOI: http://dx.doi.org/10.1145/2430475.2430478

[58] Ron C. Chiang and H. Howie Huang. 2011. TRACON: Interference-aware scheduling for data-intensive applications in virtualized environments. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[59] Ron C. Chiang, Jinho Hwang, H. Howie Huang, and Timothy Wood. 2014. Matrix: Achieving predictable virtual machine performance in the clouds. In *11th International Conference on Autonomic Computing*.

[60] Hanen Chihi, Walid Chainbi, and Khaled Ghedira. 2013. An energy-efficient self-provisioning approach for cloud resources management. *SIGOPS Oper. Syst. Rev.* 47, 3 (Nov. 2013), 2–9. DOI: http://dx.doi.org/10.1145/2553070.2553072

[61] Xabriel J. Collazo-Mojica, S. M. Sadjadi, Jorge Ejarque, and Rosa M. Badia. 2012. Cloud application resource mapping and scaling based on monitoring of QoS constraints. In *The 24th International Conference on Software Engineering and Knowledge Engineering (SEKE'12)* (Jul 2012), 88–93.

[62] Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. 2013. Multi-level elasticity control of cloud services. In *Service-Oriented Computing*, Samik Basu, Cesare Pautasso, Liang Zhang, and Xiang Fu (Eds.). Lecture Notes in Computer Science, Vol. 8274. Springer Berlin Heidelberg, 429–436. DOI: http://dx.doi.org/10.1007/978-3-642-45005-131

[63] Rodrigo da Rosa Righi, Vinicius Facco Rodrigues, Cristiano André da Costa, Guilherme Galante, Luis Carlos Erpen De Bona, and Tiago Ferreto. 2016. Autoelastic: Automatic resource elasticity for high performance applications in the cloud. *IEEE Transactions on Cloud Computing* 4, 1 (2016), 6–19.

[64] Donia El Kateb, François Fouquet, Grégory Nain, Jorge Augusto Meira, Michel Ackerman, and Yves Le Traon. 2014. Generic cloud platform multi-objective optimization leveraging models@run.time. In *29th Annual ACM Symposium on Applied Computing*. 343–350. DOI: http://dx.doi.org/10.1145/2554850.2555044

[65] Vincent C. Emeakaroha, Ivona Brandic, Michael Maurer, and Ivan Breskovic. 2011. SLA-aware application deployment and resource allocation in clouds. In *IEEE 35th Annual Computer Software and Applications Conference*. 298–303. DOI: http://dx.doi.org/10.1109/COMPSACW.2011.97

[66] Vincent C. Emeakaroha, Ivona Brandic, Michael Maurer, and Schahram Dustdar. 2010. Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In *2010 International Conference on High Performance Computing and Simulation (HPCS)*. 48–54. DOI: http://dx.doi.org/10.1109/HPCS.2010.5547150

[67] Vincent C. Emeakaroha, Rodrigo N. Calheiros, Marco A. S. Netto, Ivona Brandic, and Cesar A. F. De Rose. 2010. DeSVi: An architecture for detecting SLA violations in cloud computing infrastructures. In *2nd International ICST Conference on Cloud Computing*.

[68] Soodeh Farokhi, Pooyan Jamshidi, Ewnetu Bayuh Lakew, Ivona Brandic, and Erik Elmroth. 2016. A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach. *Future Generation Computer Systems* 65 (2016), 57–72.

[69] Hector Fernandez, Guillaume Pierre, and Thilo Kielmann. 2014. Autoscaling web applications in heterogeneous cloud infrastructures. In *2014 IEEE International Conference on Cloud Engineering (IC2E)*. 195–204. DOI: http://dx.doi.org/10.1109/IC2E.2014.25

[70] Stefano Ferretti, Vittorio Ghini, Fabio Panzieri, Michele Pellegrini, and Elisa Turrini. 2010. QoS-aware clouds. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*. 321–328. DOI: http://dx.doi.org/10.1109/CLOUD.2010.17

[71] Florian Fittkau, Sören Frey, and Wilhelm Hasselbring. 2012. CDOSim: Simulating cloud deployment options for software migration support. In *2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*. 37–46. DOI: http://dx.doi.org/10.1109/MESOCA.2012.6392599

[72] Sören Frey, Florian Fittkau, and Wilhelm Hasselbring. 2013. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *2013 IEEE International Conference on Software Engineering*. 512–521. http://dl.acm.org/citation.cfm?id=2486788.2486856

[73] Guilherme Galante and LuisCarlosErpen Bona. 2013. Constructing elastic scientific applications using elasticity primitives. In *Computational Science and Its Applications*. Lecture Notes in Computer Science, Vol. 7975. 281–294.

[74] Alessio Gambi, Giovanni Toffetti, Cesare Pautasso, and Mauro Pezze. 2013. Kriging controllers for cloud applications. *IEEE Internet Comput.* 17, 4 (July 2013), 40–47. DOI: http://dx.doi.org/10.1109/MIC.2012.142

[75] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. 2014. Adaptive, model-driven autoscaling for cloud applications. In *11th International Conference on Autonomic Computing*.

[76] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. 2017. Model-driven optimal resource scaling in cloud. *Software & Systems Modeling* (2017), 1–18.

[77] Hamoun Ghanbari, Cornel Barna, Marin Litoiu, Murray Woodside, Tao Zheng, Johnny Wong, and Gabriel Iszlai. 2011. Tracking adaptive performance models using dynamic clustering of user classes. *SIGSOFT Softw. Eng. Notes* 36, 5 (Sept. 2011), 179–188. DOI: http://dx.doi.org/10.1145/1958746.1958774

[78] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, and Gabriel Iszlai. 2011. Exploring alternative approaches to implement an elasticity policy. In *2011 IEEE International Conference on Cloud Computing (CLOUD)*. 716–723. DOI: http://dx.doi.org/10.1109/CLOUD.2011.101

[79] Sukhpal Singh Gill, Inderveer Chana, Maninder Singh, and Rajkumar Buyya. 2017. CHOPPER: An intelligent QoS-aware autonomic resource management approach for cloud computing. *Cluster Computing* (2017), 1–39.

[80] Zhenhuan Gong, Xiaohui Gu, and J. Wilkes. 2010. PRESS: PRedictive elastic resource scaling for cloud systems. In *2010 International Conference on Network and Service Management (CNSM)*. 9–16. DOI: http://dx.doi.org/10.1109/CNSM.2010.5691343

[81] Hadi Goudarzi and Massoud Pedram. 2011. Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems. In *2011 IEEE International Conference on Cloud Computing (CLOUD)*. 324–331. DOI: http://dx.doi.org/10.1109/CLOUD.2011.106

[82] Yanfei Guo, P. Lama, ChangJun Jiang, and Xiaobo Zhou. 2014. Automated and agile server parametertuning by coordinated learning and control. *IEEE Transactions on Parallel and Distributed Systems* 25, 4 (April 2014), 876–886. DOI: http://dx.doi.org/10.1109/TPDS.2013.115

[83] Rui Han, Li Guo, M. M. Ghanem, and Yike Guo. 2012. Lightweight resource scaling for cloud applications. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 644–651. DOI: http://dx.doi.org/10.1109/CCGrid.2012.52

[84] Stephen Herbein, Ayush Dusia, Aaron Landwehr, Sean McDaniel, Jose Monsalve, Yang Yang, Seetharami R. Seelam, and Michela Taufer. 2016. Resource management for running HPC applications in container clouds. In *International Conference on High Performance Computing*. Springer, 261–278.

[85] Henry Hoffman. 2013. *Seec: A Framework for Self-aware Management of Goals and Constraints in Computing Systems*. Ph.D. Dissertation. Cambridge, MA, USA. Advisor(s) Agarwal, Anant and Devadas, Srinivas.

[86] Nikolaus Huber, Fabian Brosig, and Samuel Kounev. 2011. Model-based self-adaptive resource allocation in virtualized environments. In *6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 90–99. DOI: http://dx.doi.org/10.1145/1988008.1988021

[87] IBM. 2003. An architectural blueprint for autonomic computing. *IBM Technical Report* (2003).

[88] Jing Jiang, Jie Lu, and Guangquan Zhang. 2011. An innovative self-adaptive configuration optimization system in cloud computing. In *2011 IEEE 9th International Conference on Dependable, Autonomic and Secure Computing*. 621–627. DOI: http://dx.doi.org/10.1109/DASC.2011.112

[89] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. 2013. Optimal cloud resource auto-scaling for web applications. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 58–65. DOI: http://dx.doi.org/10.1109/CCGrid.2013.73

[90] Yexi Jiang, Chang-Shing Perng, Tao Li, and R. N. Chang. 2013. Cloud analytics for capacity planning and instant vm provisioning. *IEEE Transactions on Network and Service Management* 10, 3 (September 2013), 312–325. DOI: http://dx.doi.org/10.1109/TNSM.2013.051913.120278

[91] Farhana Kabir and David Chiu. 2012. Reconciling cost and performance objectives for elastic web caches. In *2012 International Conference on Cloud and Service Computing (CSC)*. 88–95. DOI: http://dx.doi.org/10.1109/CSC.2012.21

[92] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2014. Adaptive resource provisioning for virtualized servers using kalman filters. *ACM Trans. Auton. Adapt. Syst.* 9, 2, Article 10 (July 2014), 35 pages. DOI: http://dx.doi.org/10.1145/2626290

[93] Gastón Keller, Michael Tighe, Hanan Lutfiyya, and Michael Bauer. 2013. DCSim: A data centre simulation tool. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. 1090–1091.

[94] Seoyoung Kim, Jik-Soo Kim, Soonwook Hwang, and Yoonhee Kim. 2013. An allocation and provisioning model of science cloud for high throughput computing applications. In *2013 ACM Conference on Cloud and Autonomic Computing*. Article 27, 8 pages. DOI: http://dx.doi.org/10.1145/2494621.2494649

[95] Younggyun Koh, R. Knauerhase, P. Brett, M. Bowman, Zhihua Wen, and C. Pu. 2007. An analysis of performance interference effects in virtual environments. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*. 200–209. DOI: http://dx.doi.org/10.1109/ISPASS.2007.363750

[96] George Kousiouris, Tommaso Cucinotta, and Theodora Varvarigou. 2011. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. *J. Syst. Softw.* 84, 8 (Aug. 2011), 1270–1291. DOI: http://dx.doi.org/10.1016/j.jss.2011.04.013

[97] George Kousiouris, Andreas Menychtas, Dimosthenis Kyriazis, Spyridon Gogouvitis, and Theodora Varvarigou. 2014. Dynamic, behavioral-based estimation of resource provisioning based on high-level application terms in Cloud platforms. *Future Generation Computer Systems* 32, 0 (2014), 27–40. DOI: http://dx.doi.org/10.1016/j.future.2012.05.009

[98] Sajib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta. 2012. Modeling virtualized applications using machine learning techniques. In *8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*. 3–14. DOI: http://dx.doi.org/10.1145/2151024.2151028

[99] Ewnetu Bayuh Lakew, Alessandro Vittorio Papadopoulos, Martina Maggio, Cristian Klein, and Erik Elmroth. 2017. KPI-agnostic control for fine-grained vertical elasticity. In *IEEE 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 589–598.

[100] Palden Lama, Yanfei Guo, and Xiaobo Zhou. 2013. Autonomic performance and power control for co-located Web applications on virtualized servers. In *2013 IEEE/ACM 21st International Symposium on Quality of Service*. 1–10. DOI: http://dx.doi.org/10.1109/IWQoS.2013.6550266

[101] Peter R. Lewis, Arjun Chandra, Shaun Parsons, Edward Robinson, Kyrre Glette, Rami Bahsoon, Jim Torresen, and Xin Yao. 2011. A survey of self-awareness and its application in computing systems. In *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*. 102–107. DOI: http://dx.doi.org/10.1109/SASOW.2011.25

[102] Peter R. Lewis, Arjun Chandra, Funmilade Faniyi, Kyrre Glette, Tao Chen, Rami Bahsoon, Jim Torresen, and Xin Yao. 2015. Architectural aspects of self-aware and self-expressive computing systems: From psychology to engineering. *Computer* 48, 8 (Aug 2015), 62–70. DOI: http://dx.doi.org/10.1109/MC.2015.235

[103] Hui Li, Giuliano Casale, and Tariq Ellahi. 2010. SLA-driven planning and optimization of enterprise applications. In *1st Joint WOSP/SIPEW International Conference on Performance Engineering*. 117–128. DOI: http://dx.doi.org/10.1145/1712605.1712625

[104] Jim Li, John Chinneck, Murray Woodside, Marin Litoiu, and Gabriel Iszlai. 2009. Performance model driven QoS guarantees and optimization in clouds. In *ICSE Workshop on Software Engineering Challenges of Cloud Computing*. 15–22. DOI: http://dx.doi.org/10.1109/CLOUD.2009.5071528

[105] Jim Zw Li, Murray Woodside, John Chinneck, and Marin Litoiu. 2011. CloudOpt: Multi-goal optimization of application deployments across a cloud. In *2011 7th International Conference on Network and Service Management (CNSM)*. 1–9.

[106] Li Li, Tony Tang, and Wu Chou. 2015. A rest service framework for fine-grained resource management in container-based cloud. In *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 645–652.

[107] Harold C. Lim, Shivnath Babu, Jeffrey S. Chase, and Sujay S. Parekh. 2009. Automated control in cloud computing: Challenges and opportunities. In *1st Workshop on Automated Control for Datacenters and Clouds*. 13–18. DOI: http://dx.doi.org/10.1145/1555271.1555275

[108] Wes Lloyd, Shrideep Pallickara, Olaf David, Jim Lyon, Mazdak Arabi, and Ken Rojas. 2012. Performance modeling to support multi-tier application deployment to infrastructure-as-a-service clouds. In *2012 IEEE 5th International Conference on Utility and Cloud Computing (UCC)*. 73–80. DOI: http://dx.doi.org/10.1109/UCC.2012.20

[109] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. 2014. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing* 12, 4 (2014), 559–592.

[110] Hongyi Ma, Liqiang Wang, Byung Chul Tak, Long Wang, and Chunqiang Tang. 2016. Auto-tuning performance of MPI parallel programs using resource management in container-based virtual cloud. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 545–552.

[111] Amiya K. Maji, Subrata Mitra, Bowen Zhou, Saurabh Bagchi, and Akshat Verma. 2014. Mitigating interference in cloud services by middleware reconfiguration. In *15th International Conference on Middleware*. 277–288. DOI: http://dx.doi.org/10.1145/2663165.2663330

[112] Zoltán Ádám Mann. 2015. Allocation of virtual machines in cloud data centers?a survey of problem models and optimization algorithms. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 11.

[113] Sunilkumar S. Manvi and Gopal Krishna Shyam. 2014. Resource management for infrastructure as a service (IaaS) in cloud computing: A survey. *Journal of Network and Computer Applications* 41 (2014), 424–440.

[114] Michael Maurer, Ivona Brandic, Vincent C. Emeakaroha, and Schahram Dustdar. 2010. Towards knowledge management in self-adaptable clouds. In *2010 6th World Congress on Services (SERVICES-1)*. 527–534. DOI: http://dx.doi.org/10.1109/SERVICES.2010.26

[115] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. 2012. Self-adaptive and resource-efficient SLA enactment for cloud computing infrastructures. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. 368–375. DOI: http://dx.doi.org/10.1109/CLOUD.2012.55

[116] Aly Megahed, Mohamed Mohamed, and Samir Tata. 2017. A stochastic optimization approach for cloud elasticity. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 456–463.

[117] Rizwan Mian, Patrick Martin, Farhana Zulkernine, and Jose Luis Vazquez-Poletti. 2013. Towards building performance models for data-intensive workloads in public clouds. In *4th ACM/SPEC International Conference on Performance Engineering*. 259–270. DOI: http://dx.doi.org/10.1145/2479871.2479908

[118] Dorian Minarolli and Bernd Freisleben. 2013. Virtual machine resource allocation in cloud computing via multi-agent fuzzy control. In *2013 Third International Conference on Cloud and Green Computing (CGC)*. 188–194. DOI: http://dx.doi.org/10.1109/CGC.2013.35

[119] Dorian Minarolli and Bernd Freisleben. 2014. Distributed resource allocation to virtual machines via artificial neural networks. In *22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing*. 490–499. DOI: http://dx.doi.org/10.1109/PDP.2014.102

[120] Vladimir Nikolov, Steffen Kächele, Franz J. Hauck, and Dieter Rautenbach. 2014. CLOUDFARM: An elastic cloud platform with flexible and adaptive resource management. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*. 547–553. DOI: http://dx.doi.org/10.1109/UCC.2014.84

[121] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. 2009. Automated control of multiple virtualized resources. In *4th ACM European Conference on Computer Systems*. 13–26. DOI: http://dx.doi.org/10.1145/1519065.1519068

[122] Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. 2017. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys* 9, 4 (2017), Article 39, 34 pages.

[123] Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. 2016. A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. *Journal of Network and Computer Applications* 65 (2016), 167–180.

[124] Navaneeth Rameshan, Ying Liu, Leandro Navarro, and Vladimir Vlassov. 2016. Augmenting elasticity controllers for improved accuracy. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 117–126.

[125] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, and Kun Wang. 2011. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In *2011 IEEE 19th International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*. 45–54. DOI: http://dx.doi.org/10.1109/MASCOTS.2011.47

[126] Jia Rao, Yudi Wei, Jiayu Gong, and Cheng-Zhong Xu. 2011. DynaQoS: Model-free self-tuning fuzzy control of virtualized resources for QoS provisioning. In *2011 IEEE 19th International Workshop on Quality of Service (IWQoS)*. 1–9. DOI: http://dx.doi.org/10.1109/IWQOS.2011.5931341

[127] Nathuji Ripal, Kansal Aman, and Ghaffarkhah Alireza. 2010. Q-clouds: Managing performance interference effects for QoS-aware clouds. In *5th European Conference on Computer Systems*. 237–250. DOI: http://dx.doi.org/10.1145/1755913.1755938

[128] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.* 4, 2, Article 14 (May 2009), 42 pages. DOI: http://dx.doi.org/10.1145/1516533.1516538

[129] Mina Sedaghat, Francisco Hernandez-Rodriguez, and Erik Elmroth. 2013. A virtual machine re-packing approach to the horizontal vs. vertical elasticity tradeoff for cloud autoscaling. In *2013 ACM Conference on Cloud and Autonomic Computing*. Article 6, 10 pages. DOI: http://dx.doi.org/10.1145/2494621.2494628

[130] Seetharami R. Seelam, Paolo Dettori, Peter Westerink, and Ben Bo Yang. 2015. Polyglot application auto scaling service for platform as a service cloud. In *2015 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 84–91.

[131] R. S. Shariffdeen, D. T. S. P. Munasinghe, H. S. Bhathiya, U. K. J. U. Bandara, and H. M. N. Dilum Bandara. 2016. Workload and resource aware proactive auto-scaler for paas cloud. In *2016 IEEE 9th International Conference on Cloud Computing*. 11–18.

[132] Upendra Sharma, P. Shenoy, S. Sahu, and A. Shaikh. 2011. A cost-aware elasticity provisioning system for the cloud. In *2011 31st International Conference on Distributed Computing Systems (ICDCS)*. 559–570. DOI: http://dx.doi.org/10.1109/ICDCS.2011.59

[133] Andre Abrantes D. P. Souza and Marco A. S. Netto. 2015. Using application data for sla-aware auto-scaling in cloud environments. In *2015 IEEE 23rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 252–255.

[134] Yu Sun, Jules White, Sean Eade, and Douglas C. Schmidt. 2016. ROAR: A QoS-oriented modeling framework for automated cloud resource allocation and optimization. *Journal of Systems and Software* 116 (2016), 146–161.

[135] Yu Sun, Jules White, Bo Li, Michael Walker, and Hamilton Turner. 2017. Automated QoS-oriented cloud resource optimization using containers. *Automated Software Engineering* 24, 1 (2017), 101–137.

[136] Vincent van Beek, Jesse Donkervliet, Tim Hegeman, Stefan Hugtenburg, and Alexandru Iosup. 2015. Mnemos: Self-expressive management of business-critical workloads in virtualized datacenters. (2015).

[137] Christian Vecchiola, Xingchen Chu, and Rajkumar Buyya. 2009. Aneka: A software platform for .NET-based cloud computing. *High Speed and Large Scale Scientific Computing* 18 (2009), 267–295.

[138] Hiroshi Wada, Junichi Suzuki, Yuji Yamano, and Katsuya Oba. 2011. Evolutionary deployment optimization for service-oriented clouds. *Softw. Pract. Exper.* 41, 5 (April 2011), 469–493. DOI: http://dx.doi.org/10.1002/spe.1032

[139] Chen Wang, Junliang Chen, Bing Bing Zhou, and A. Y. Zomaya. 2012. Just satisfactory resource provisioning for parallel applications in the cloud. In *2012 IEEE Eighth World Congress on Services (SERVICES)*. 285–292. DOI: http://dx.doi.org/10.1109/SERVICES.2012.38

[140] Lixi Wang, Jing Xu, and Ming Zhao. 2012. Application-aware cross-layer virtual machine resource management. In *ACM 9th International Conference on Autonomic Computing*. 13–22. DOI: http://dx.doi.org/10.1145/2371536.2371541

[141] Lixi Wang, Jing Xu, and Ming Zhao. 2012. Tracking adaptive performance models using dynamic clustering of user classes. In *7th International Workshop on Feedback Computing*.

[142] Bhathiya Wickremasinghe, Rodrigo N. Calheiros, and Rajkumar Buyya. 2010. CloudAnalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *2010 24th IEEE International*

Conference on Advanced Information Networking and Applications (AINA). 446–452. DOI: http://dx.doi.org/10.1109/AINA.2010.32

[143] Fetahi Wuhib, Rolf Stadler, and Hans Lindgren. 2012. Dynamic resource allocation with management objectives: Implementation for an OpenStack cloud. In *2012 8th International Conference and 2012 Workshop on Systems Virtualiztion Management Network and Service Management (Cnsm), (svm)*. 309–315.

[144] Pengcheng Xiong, Yun Chi, Shenghuo Zhu, Hyun Jin Moon, C. Pu, and H. Hacgumus. 2015. SmartSLA: Cost-sensitive management of virtualized resources for CPU-bound database services. *IEEE Transactions on Parallel and Distributed Systems* 26, 5 (2015), 1441–1451. DOI: http://dx.doi.org/10.1109/TPDS.2014.2319095

[145] Pengcheng Xiong, Calton Pu, Xiaoyun Zhu, and Rean Griffith. 2013. vPerfGuard: An automated model-driven framework for application performance diagnosis in consolidated cloud environments. In *4th ACM/SPEC International Conference on Performance Engineering*. 271–282. DOI: http://dx.doi.org/10.1145/2479871.2479909

[146] Pengcheng Xiong, Zhikui Wang, S. Malkowski, Qingyang Wang, D. Jayasinghe, and C. Pu. 2011. Economical and robust provisioning of N-Tier cloud workloads: A multi-level control approach. In *2011 31st International Conference on Distributed Computing Systems (ICDCS)*. 571–580. DOI: http://dx.doi.org/10.1109/ICDCS.2011.88

[147] Cheng-Zhong Xu, Jia Rao, and Xiangping Bu. 2012. URL: A unified reinforcement learning approach for autonomic cloud management. *J. Parallel and Distrib. Comput.* 72, 2 (2012), 95–105. DOI: http://dx.doi.org/10.1016/j.jpdc.2011.10.003

[148] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. 2014. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers* 16, 1 (2014), 7–18. DOI: http://dx.doi.org/10.1007/s10796-013-9459-0

[149] Lenar Yazdanov and Christof Fetzer. 2013. VScaler: Autonomic virtual machine scaling. In *2013 IEEE 6th International Conference on Cloud Computing*. 212–219. DOI: http://dx.doi.org/10.1109/CLOUD.2013.142

[150] Haitao Zhang, Huadong Ma, Guangping Fu, Xianda Yang, Zhe Jiang, and Yangyang Gao. 2016. Container based video surveillance cloud service with fine-grained resource provisioning. In *IEEE 9th International Conference on Cloud Computing*. 758–765.

[151] Qi Zhang, Quanyan Zhu, and R. Boutaba. 2011. Dynamic resource allocation for spot markets in cloud computing environments. In *2011 4th IEEE International Conference on Utility and Cloud Computing (UCC)*. 178–185. DOI: http://dx.doi.org/10.1109/UCC.2011.33

[152] Ying Zhang, Gang Huang, Xuanzhe Liu, and Hong Mei. 2010. Integrating resource consumption and allocation for infrastructure resources on-demand. In *2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*. 75–82. DOI: http://dx.doi.org/10.1109/CLOUD.2010.11

[153] Tao Zheng, Marin Litoiu, and Murray Woodside. 2011. Integrated estimation and tracking of performance model parameters with autoregressive trends. In *ACM/SPEC International Conference on Performance Engineering*. 157–166. DOI: http://dx.doi.org/10.1145/1958746.1958772

[154] Qian Zhu and G. Agrawal. 2012. Resource provisioning with budget constraints for adaptive applications in cloud environments. *IEEE Transactions on Services Computing* 5, 4 (Fourth 2012), 497–511. DOI: http://dx.doi.org/10.1109/TSC.2011.61

[155] Zhiliang Zhu, Jing Bi, Haitao Yuan, and Ying Chen. 2011. SLA based dynamic virtualized resources provisioning for shared cloud data centers. In *2011 IEEE International Conference on Cloud Computing (CLOUD)*. 630–637. DOI: http://dx.doi.org/10.1109/CLOUD.2011.91

Received April 2017; revised January 2018; accepted February 2018