Department: Internet of Things, People, and Processes
Editor: Schahram Dustdar, dustdar@dsg.tuwien.ac.at

SLOC: Service Level Objectives for Next Generation Cloud Computing

Stefan Nastic
Distributed Systems Group, TU Wien

Andrea Morichetta
Distributed Systems Group, TU Wien

Thomas Pusztai
Distributed Systems Group, TU Wien

Schahram Dustdar
Distributed Systems Group, TU Wien

Xiaoning Ding
Futurewei Technologies, Inc.

Deepak Vij
Futurewei Technologies, Inc.

Ying Xiong
Futurewei Technologies, Inc.

***Abstract*—Since the emergence of cloud computing service level objectives (SLOs) and service level agreements (SLAs) have put themselves forward as one of the key enablers for cloud's on-demand, pay-as-you-go service consumption model. To date, the vast majority of cloud platforms provide support for SLAs only in terms of statically predefined SLOs, e.g., service availability, and low-level resource capacity guarantees, e.g., CPU usage. Unfortunately, there is only limited support to clearly map workload performance requirements to the resource capacity guarantees. In this article, we introduce SLOC— a novel elasticity framework, which promotes a novel performance-driven, SLO-native approach to cloud computing. We outline the main research challenges, vision, and approach of our SLOC framework toward the SLO-native paradigm in next generation cloud computing.**

Digital Object Identifier 10.1109/MIC.2020.2987739
Date of current version 21 July 2020.

Cloud's service consumption model inherently requires a formal agreement between a customer (service consumer) and a cloud service provider. This is usually described as service level agreement (SLA).[2,17] Key components of any SLA are service level objectives (SLOs). They define specific and measurable capacity guarantees to a customer, e.g., available memory to a provisioned VM. However, customers are usually more interested in performance guarantees, whose impact can be reflected in business key performance indicators (KPIs). Unfortunately, currently there is only limited support to clearly map the resource capacity requirements to the guarantees about workload performance. Defining application SLOs is largely performed on an ad hoc basis, with limited support from cloud providers. This poses a significant challenge to customers as it is usually very difficult to correctly derive low-level resource capacity requirements, such as memory allocation, from a workload's business requirements.

Elasticity, one of the fundamental properties of cloud computing, allows for applications to respond to varying load patterns by adjusting the amount of provisioned resources to exactly match their current need, thus minimizing over-provisioning and reducing hosting costs. Although, elasticity offers theoretically ideal strategies to keep track of and enforce an application's resource-bound SLOs, in practice it is increasingly complex to define correct elastic strategies in the face of ever-changing cloud services, novel and heterogeneous resource types (e.g., containers), execution paradigms (e.g., serverless computing), and deployments topologies (e.g., microservice meshes). In addition to resource elasticity, the cloud computing paradigm enables additional elasticity dimensions, such as cost elasticity and quality elasticity.[7] However, current approaches dealing with SLOs largely fail to consider elasticity dimensions holistically. This introduces another level of indirection for the users, who now need to consider not only implications of low-level resources on, for example, costs, but also implications of costs on business KPIs.

SLOs also play a crucial role for cloud service providers, as their main business model depends on achieving the best utilization of their resource pools, while maintaining a minimal level of SLA violations. Although, cloud service providers employ numerous techniques to optimize resource utilization in their data centers, such as resource overcommitment and oversubscription models,[12] resource bursting,[22] and resources reclamations,[8] the average utilization of data centers' resource pools is still reported to be as low as 20%–30%. This has a negative impact on both the cloud service providers and the customers, because customers are usually over provisioning for their workloads, hence incurring higher operational costs, while cloud service providers are left with de facto unused resources, hence are not able to fully exploit their economies of scale.

In this article, we introduce SLOC—a novel elasticity framework that promotes SLOs as a first-class cloud citizen. SLOC aims to raise the level of abstraction and provide suitable models, algorithms, runtime mechanisms, and tools for providing and consuming cloud resources in an SLO-native manner, while at the same time offering performance guarantees to the users. Concretely, SLOC framework intends to relieve users from explicitly dealing with low-level concepts such as CPU and memory, by enabling *performance-driven, SLO-native approach* to cloud computing. At the same time, framework aims to enable cloud providers to have more versatile understanding, greater control, and better utilization of their resource pools by, among other things, accounting for additional elasticity dimensions such as cost and quality.

The remainder of this article is structured as follows. In the "Research Challenges" section, we analyze key research challenges, which motivate our SLOC framework. In the "Background" section, we provide an overview of our previous work in the field of elastic computing, SLAs, and cloud engineering. The "SLOC Approach Overview" section presents the main vision and introduces the general approach of SLOC framework. In the "Design Considerations for SLO-Driven Elasticity Mechanisms" section, we discuss main techniques, mechanisms, and the road map for our SLOC framework. Related research is discussed in the "Related Work" section. Finally,

we conclude this article in the "Conclusion" section and provide an outlook for ongoing and future work.

RESEARCH CHALLENGES

RC-1: Specifying multidimensional elasticity SLOs: The vast majority of cloud platforms allow for specifying resource guarantees, but very few offer any support for additional elasticity dimensions, such as cost or quality. This significantly limits the end user (consumer), who might care more to optimize for the costs of executing a specific workload. Unfortunately, apart from some niche solutions (e.g., AWS Spot Instances) there is very limited support for multidimensional elasticity concerns and optimization.

RC-2: Mapping performance SLOs to resource allocation: Currently, most cloud platforms provide rudimentary support for specifying QoS and SLOs to end users. The support is mainly limited to statically predefined SLAs (e.g., service availability) and low-level resource capacity guarantees (e.g., requests/limits in Kubernetes or templates/types in AWS). Unfortunately, correctly mapping the desired workload's performance models to resource capacities, as well as correlating the two metric sets and deriving their implications, is a very challenging task in practice. More concretely, in our case, this entails automatically identifying the underlying cloud resources to be allocated/adjusted in accordance with the interdependent three-dimensional elasticity model.

RC-3: Supporting full-stack observability for SLO tracking: Observability is a foundational element for building and running modern cloud-native and microservice applications. However, currently, there is only limited support for managing QoS and SLO performance models at the level of logical deployment units and topologies. Additionally, support for aggregating low-level resource capacity requirements and metrics across the full-stack is still largely missing. This is crucial for deriving macro-level metrics such as average CPU consumption of a replica set level. Therefore, there are a number of challenges that development and operations teams face when dealing with observability.

RC-4: Enabling consistent and congruent scaling strategies: While there are numerous tools supporting infrastructure provisioning and configuration management, support for run time elastic controlling remains fairly limited. It is primarily based on a notion of autoscaling groups/sets of resources where a user specifies their cardinality. Most of the solutions only allow for service- or instance-level scaling policies as opposed to full-stack scaling strategies. On the one hand, this makes dealing with elasticity concerns cumbersome. On the other hand, synchronizing scaling actions across multiple services and defining consistent and congruent scaling strategies become very difficult. Finally, optimizing scaling actions for specific elasticity and SLO dimensions, such as cost, have only a rudimentary support. Therefore, creating suitable elastic scaling strategies to date remains a challenging task.

BACKGROUND

This article builds on our previous research in the fields of elastic computing, SLAs, and cloud engineering. In continuation, we outline the most important aspects of our previous work underpinning our SLOC framework.

Elastic Computing—The Next Level

Each elasticity dimension has different indicators or metrics that contribute to it. The minimum and maximum allowed values of a set of metrics define an *elasticity boundary*. The lower boundary indicates, in most cases, the minimum resources and quality characteristics required by an application/component to fulfill its purpose, while the upper boundary usually describes the maximum that the developer can afford to pay. In some cases, e.g., the response time, the upper boundary refers to the minimum requirement. When the metrics of all dimensions are combined, they form an application's *elasticity space*. This is essentially the current state of the application with respect to its elasticity metrics (e.g., current response time, cost per hour, etc.). It can be tracked over time to understand the behavior of the application and the relationships between various metrics.[4,7,20,19] MELA[20] is a tool for advanced monitoring of cloud systems. It collects metrics from various levels of a cloud system (single services, VMs, etc.) and aggregates and

combines them, according to the developer's configuration, to derive higher level metrics. Rather than focusing on a single component of a cloud application, it thus delivers insights into the current elasticity space of the entire application. By analyzing subsets of the metrics over time, it furthermore determines *elasticity pathways*, which shows the relationships among a set of metrics over time.

SYBL[4] is a language for specifying elasticity monitoring, constraints, and strategies at different levels of cloud applications. Specifically, these levels are the entire application, application components, and code within the components. Based on these specifics, the runtime environment can apply complex elasticity control actions on the application.

Cloud Platforms

Resource scheduling is one of the fundamental properties in the current cloud platforms. In this article, we leverage our previous work, Arktos,[9] as the cloud platform to build and experiment multidimensional elastic computing, i.e., extending resource elasticity to include quality and cost elasticity dimensions. Arktos is designed as a multitenant and large scale cloud platform evolved from Kubernetes[1] to schedule, provision, and manage resources for VM, container, and serverless functions.

Specifically, the vision of Arktos is to 1) manage a very large compute cluster, in the order of 100K compute nodes, this allows us to have the scale to experiment various scheduling algorithms for increasing resource utilization for cloud providers; 2) unify the technology stack to manage various resource types such as bare metal servers, virtual machines, and lightweight containers, which allows the platform to manage resources more efficiently at different granularity level, thus achieving the true elasticity of the platform; 3) provide true multitenant computing environment with strong isolation so that resources can be shared without impacting other tenants. This further improves resource utilization by using techniques such as resource reclamation among tenants.

Another of our projects, which underpins this article, is Mizar[10] cloud network. Arktos uses Mizar as its underlying network control and data plane to provision and manages the virtual network and network endpoints for VMs and containers provisioned by Arktos. Mizar provides support of large scale network endpoint provisioning, fast endpoint-to-endpoint network routing among VMs, and containers.

SLOC APPROACH OVERVIEW

Main Objectives

To address the aforementioned challenges and achieve our vision of an SLO-native paradigm for the next generation cloud platforms, our SLOC framework sets the following main objectives.

- We intend to exploit and advance current support for managing elasticity concerns in order to achieve better support for SLOs. To this end, we intend to build on our previous work on elasticity space and boundaries and to facilitate defining and enforcing *soft and hard SLO constraints in terms of elasticity mechanisms*.
- We aim to enable the users to define their SLO requirements in a *flexible and use-case-specific manner*. Opposed to current SLA support, which only accounts for general, predefined requirements such as availability or durability, our objective is to go a step further and facilitate optimizing on a use case basis. For example, for some nonmission-critical jobs and services, the infrastructure costs might be the main factor driving the decisions.
- To be able to define and specify cloud-native SLAs in terms of *business-relevant, performance-based SLOs*, we intend to develop a novel SLO elasticity policy language. The language aims to allow users to perform a clear *mapping of low-level resource quotas to a workload's performance models* and to enable defining complex scaling strategies.
- In order to enable the shift from low-level, resource-centered SLOs and elastic requirement specifics towards *intent-focused, multidimensional elasticity models*, our framework will provide novel coordination, control, and orchestration approaches that enable cloud platforms to adapt dynamically to varying load patterns in a dependable manner.

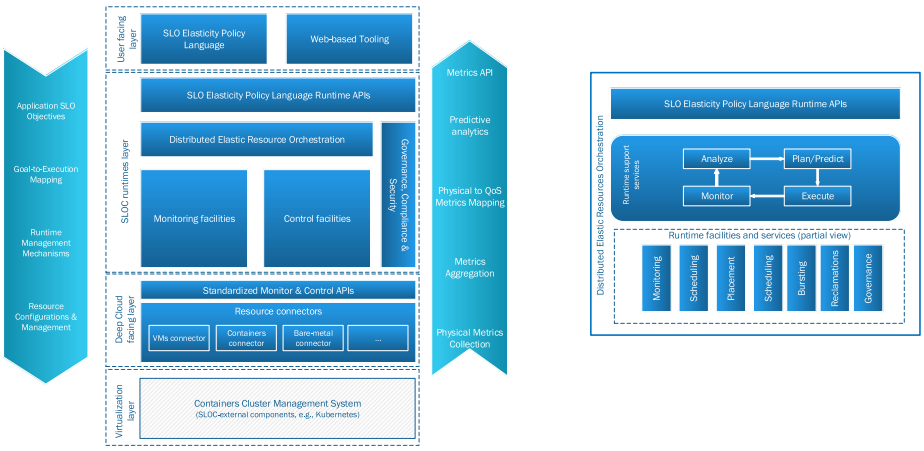

Figure 1. Overview of SLOC's architecture and distributed elastic resource orchestration.

In the remainder of this section, we describe the main concepts and design of the SLOC framework, which enables us to achieve the aforementioned objectives. In the "Design Considerations for SLO-Driven Elasticity Mechanisms" section, we discuss the main SLO-driven elasticity mechanisms and techniques w.r.t. our general objectives.

Main Concepts and Architecture Overview

The core idea behind SLOC framework is to enable SLO-native management of elastic Cloud resources. In a nutshell, our *SLO-native approach* introduces a paradigm shift from general, business logic agnostic, low-level SLAs to *intent-based, SLO-first, performance-driven, and orchestration-aware* elasticity models.

1. Intent-based means that we are declaring what the execution environment of a workload should look like, i.e., the infrastructure/resource desired state for the given point in time considering the elasticity requirements, constraints, and SLAs.
2. SLO-first means that all the SLOs and QoS along multiple elasticity dimensions are inherently aligned with application model and business requirements/KPIs as opposed to external, predefined, business logic agnostic configuration
3. Performance driven denotes that the SLOs' boundaries are specific in terms of high-level models, which represent workload's desired performance requirements. In turn, such performance models are described in form of SLO or QoS metrics such as latency, timeout rate queue size, etc., as opposed to physical resource consumption metrics such as CPU and memory.
4. Orchestration-aware implies that the application deployment bundles such as microservices or service meshes are aware of automated deployment, scaling, scheduling, and management.

Subsequently, we outline the architecture of our SLOC framework and give an overview of its main components. Figure 1 (left) shows a high-level view of the framework architecture together with main control processes (top-down) and monitoring data delivery and analytics process (bottom-up). On a high-level, we identify the following three main layers of SLOC framework.

i) User-facing layer.
ii) SLOC runtimes layer.
iii) Deep Cloud facing layer.

The user-facing layer exposes main abstractions, mechanisms, and runtime services of the SLOC framework to the end users, enabling them to define SLO elastic policies, visualize their infrastructure and its current status, as determined by SLOC's observability mechanisms. We discuss SLOC's SLO Elastic Policy Language in detail in the "Design Considerations for SLO-Driven Elasticity Mechanisms" section.

The SLOC runtimes layer is the "brains" of our framework. Generally, it is responsible for implementing main models, algorithms, and mechanisms, which are required to interpret

user-defined SLOs, translates such SLOs and QoS requirements in cloud-specific resource provisioning models and enforce such SLOs during runtime. The most important components at this layer include: i) Elastic resource orchestration. ii) Observability facilities. iii) Control facilities. iv) Governance, compliance, and security concerns. The elastic resource orchestration is responsible for interpreting and enforcing user-defined SLOs, elasticity requirements, and configuration models. Figure 1 (right) illustrates the main facilities and services of SLOC's distributed elastic resource orchestration layer. This layer acts as a "gluing" component bringing together SLO definitions, multidimensional elasticity models, and framework's runtime mechanisms. For example, elastic resource orchestration receives policy configuration directives, in terms of high-level objectives such as to optimize infrastructure for latency. It interprets these objectives and orchestrates the underlying resources accordingly, by invoking the underlying runtime mechanisms as well as delegating specific decisions and/or responsibilities to cluster management system controllers. The details of observability and control facilities are discussed in the "Design Considerations for SLO-Driven Elasticity Mechanisms" section.

SLOC's third core layer is the Deep Cloud facing layer. This layer is responsible for abstracting the underlying infrastructure resources and to mediate interaction with the cluster management system, cloud platform, and virtualization providers. Generally, its main purpose is to capture infrastructure-specific functionality and models, hence enabling development of generic SLOC runtime mechanisms, which are independent from platform implementation or vendors. The Deep Cloud facing layer will be implemented as a set of plugins, enabling SLOC framework to be easily extended as well as configured and optimized for infrastructure-specific capabilities. This is in line with SLOC's design philosophy not to "reinvent the wheel," but rather to align itself with de facto standards, such as Kubernetes, and reuse available elasticity mechanisms (e.g., for container/VM scheduling, placement mechanisms, etc.). Additionally, this approach enables the users to implement custom or even hardware-specific connectors to support, for example, different types of GPUs in the cluster.

DESIGN CONSIDERATIONS FOR SLO-DRIVEN ELASTICITY MECHANISMS

SLO Elasticity Policy Language

In order to define SLOs in a user-friendly manner, an appropriate language is required. To this end, we intend to extend the previously mentioned SYBL language. In its current state, SYBL already allows specifying constraints for all three elasticity dimensions (resources, cost, and quality).

The new SLO elasticity policy language will allow developers to define SLOs by referring to their workloads and their components as entities, for which they can define constraints on certain characteristics, e.g., the response time. It will be possible for developers to define their own entities by combining existing ones. For example, the nodes of an Apache Cassandra DB and those of an Apache Hadoop cluster could be combined in a QueryService entity, for which the developer may want to define requirements. Listing 1 shows what a simple specification in this language could look like.

Listing 1: SLO Elasticity Policy Language

```
# SLOs that apply to the entire application
# consisting of multiple containers and/or VMs.
SLO1: App.quality.responseTime < 200 ms
SLO2: App.cost.totalCost < 800 Euro

# Define a component by combining Cassandra
    and Hadoop nodes.
Comp: QueryService = [Deployments.Cassandra-
    ClusterA, Deployments.HadoopClusterB]

# This SLO applies only to the QueryService
    component.
SLO3: QueryService.cost.costEffectiveness = 350

# SLO2 is a soft SLO.
Soft: SLO2
```

Defining a response time and maximal total costs, e.g., SLO1 and SLO2 should be enough, i.e., developers should not have to worry about CPU or memory. The set of confinable characteristics provided by the language needs to be carefully devised, taking the entire SLOC framework into account. High-level specifications must be possible, but it must also be possible to map these to resources and actions for orchestration. A

characteristic can thus be included in the language if it is i) relevant for expressing a requirement posed by developers and ii) can be mapped (by itself or in conjunction with other characteristics) to low-level resource requirements or corrective actions if it is not fulfilled in a state of the running system. For example, SLO3 defines a cost effectiveness for the QueryService. The effectiveness could be influenced by characteristics, such as accuracy of the output and response time, while the cost relates to the total cost of the resources (CPU cores, memory, etc.) required by this component. The addition of custom characteristics will also be supported. If an application requires combining, e.g., CPU utilization, memory usage, and disk I/O operations, in a use case specific high-level characteristic, developers can write an extension library for this and expose it in the elasticity policy language.

Finally, for application-level cases, we can introduce a notion of soft SLO constraints, which are an extension of the traditional SLOs and our elasticity boundaries and elasticity pathway-models. These soft constraints can serve as "guidelines" to the SLOC framework on how to optimize the underlying infrastructure and navigate the application through its elasticity boundaries during runtime. For example, a total cost of less than 800 Euros is desirable (SLO2 is a soft SLO), but it is not a strict limit. On the other hand, the maximum response time of 200 ms needs to hold all the time, e.g., because the workload is part of a real-time service, thus making this a hard SLO. Thus, the "softness" of soft SLOs needs to be precisely define when designing the elasticity policy language and its runtime and possibly even be configurable by the developers.

SLOC Observability

Observability is essential to present a detailed description of the workload's behavior, providing rich contextual information. The classic observability concept is based on three ground data sources, namely metrics-based monitoring, that alert when something happened, log, that collect the event history, and traces that point to where the event locates. Observability aims at looking at what caused a specific behavior, combining all the data collected above, thus extracting relevant and appropriate information.

SLOC aims at expanding the compass, supporting detailed insights into and visualization of the workload's behavior during runtime. We intend to connect distributed traces, logs, and monitoring metrics through independent and intelligent analysis of the elastic cloud. Predictions are the result of the use of machine learning and statistical approaches, thus moving toward an automated and autonomic approach from a heavily human-centered analysis. The constant monitoring of the applications will allow precise and insightful models that can be used to predict changes in performance and proactively act on that, as well as providing a better comprehension of the correlation among individual metrics.

Furthermore, as previously discussed, SLOC aims to enable users to specify high-level performance-oriented QoS/SLOs for their workloads, which in turn get mapped to low-level resource metrics. Figure 2 depicts a partial view of SLOC's metrics taxonomy that facilitates this mapping. From collecting the resource utilization measurements, it is possible to group the extracted metrics, classifying them according to their function. For example, in Figure 2, we specify four low-level metrics classes: computation, memory, network, and storage. The high-level metrics such as Cost-Effectiveness can be mapped to such "physical metrics" by our framework automatically. Additionally, SLOC will provide mechanisms and techniques, enabling the users to also provide their custom metrics and the desired mappings in order to be able to account for application-specific behavior and concerns.

In this way, it is possible to guarantee at runtime the verification of multidimensional elasticity and SLOs models, through ensuring constraints abidance and alarming. The insights are more personalized, uncovering hidden patterns, and allowing more proactive and automated system management.

SLOC Elasticity Control

The elasticity controlling mechanisms and technique encapsulate actuation functionality of SLOC's distributed elastic resource orchestration. Figure 1 (right) illustrates a number of runtime facilities and services. Scheduling, placement, bursting, and reclamation are all examples of different elastic controls. As many of the needed

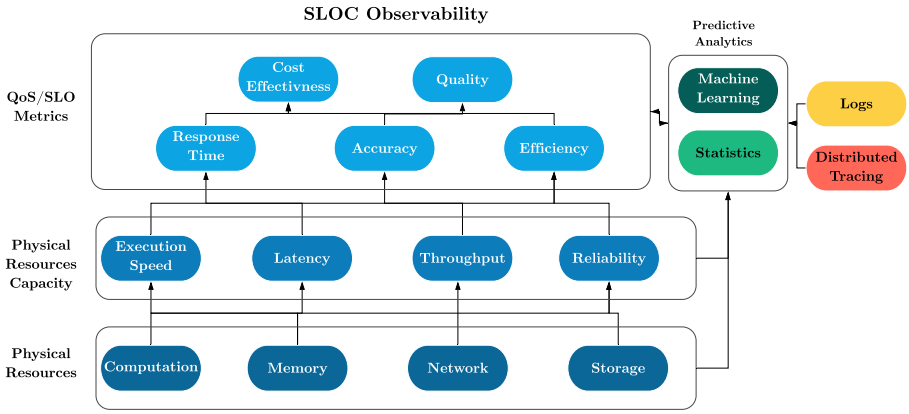

Figure 2. Example of metric taxonomy from low-level resource metrics to performance QoS/SLO metrics.

runtime services are readily implemented by cluster management system controllers, SLOC takes a pragmatic approach and delegates specific tasks to these controllers whenever possible. However, SLOC adds fundamental improvements to managing elasticity concerns in an SLO-native manner.

SLOC aims to support a set of generic, atomic scaling actions, and an extensible plugins architecture. To this end, we plan to examine and support most promising, de facto standard resource management and scaling actions. These actions are referred to as atomic scaling actions and include the following.

i) Increase/decrease resource allocations to workload/job replica (vertical scaling).
ii) Add/remove workload replica to different node, e.g., spin up a new container (horizontal scaling).
iii) Migrate a workload replica (live migration).

SLOC intends to enable applying these actions uniformly regardless of the underlying resource type (e.g., container or VM) and to build on these actions combining them into complex scaling strategies, which operate on the level deployment units. Finally, as the amount of potential scaling strategies is unlimited, we will not attempt to support every possible strategy, but rather enable easy configuration and plug-in mechanisms for SLOC framework.

SLOC intends to enable defining consistent and congruent scaling strategies. In a nutshell, this requires aligning and synchronizing scaling actions across multiple services comprising a deployment unit. A deployment unit is any set of microservices that are deployed and managed together. A set of replicated API services together with their load balancer is a basic example of a deployment unit. Therefore, scaling such deployment unit goes beyond traditional atomic scaling actions and also requires considering and dynamically changing its deployment topology. This approach enables dealing with elasticity on a higher levels of abstraction, but also exposes additional information pertinent to optimal resource management. In the above example, we can consider load balancer's queue size to optimally apply scaling actions. Additionally, it allows for optimizing scaling strategies for various elasticity and SLO dimensions, such as cost.

In addition to predictive monitoring, SLOC aims to provide support for predictive scaling. In a nutshell, SLOC will try to continuously perform sequences of prophylactic scaling actions (preventing potential SLO violations), instead of solely focusing to exactly predict the workload's future behavior. The main rationale behind this approach is utilizing discrete and deterministic nature of the scaling actions. To this end, we aim to develop several scaling strategy blueprints, which will encapsulate common scaling patterns. SLOC's resource orchestrator will then continuously assess the outcomes of applying the scaling blueprints (dry run) and act by applying them when conditions are met.

Implementation Considerations

The SLOC framework aims to build on and extend our previous work: Arktos, MELA, and

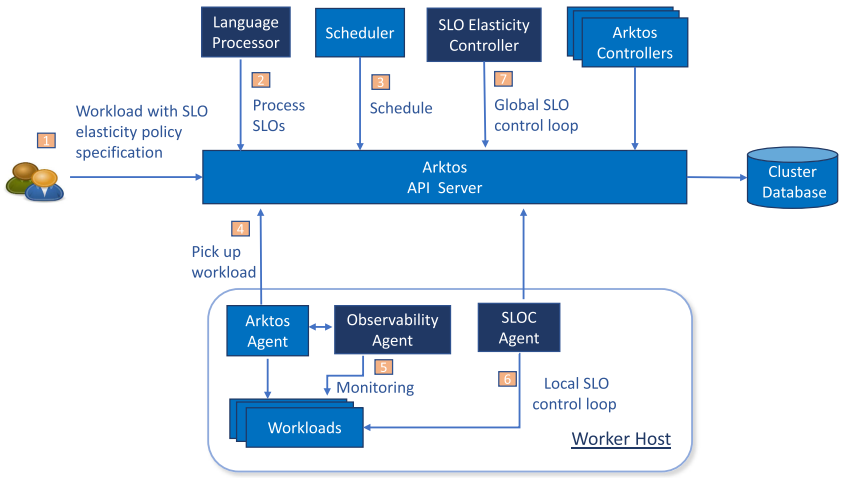

Figure 3. SLOC component diagram (partial view).

SYBL. These three systems complement each other and will work together to support implementing the SLOC architecture as described in the "SLOC Approach Overview" section.

Figure 3 illustrates the general workflow of how we envision to combine these systems and extend them in order to implement SLOC's end-to-end SLO and elasticity specific monitoring and enforcement.

1. A user submits a workload to Arktos. As a part of the workload definition, an SLOC policy object described by SLOC's policy language, which is based on SYBL, is included.
2. An SLOC language processor will do syntax and semantics check on the submitted SLOC policy, and generates preliminary resource allocation guidelines for Arktos controllers such as scheduler to optimize initial workload placement.
3. The workload is scheduled to a worker host.
4. The Arktos agent on the worker host picks up the workload and launches it.
5. An observability agent, based on MELA and Prometheus is deployed on each Arktos slave node. It monitors the workload execution and provides full-stack monitoring and observability capabilities.
6. An SLOC agent is deployed on each slave node. It detects and predicts any potential SLO violation, and performs enforcement actions that are local to a node. Such actions include increasing or decreasing resources to local workload replica (vertically scaling).
7. An SLOC controller is deployed per cluster. This controller is responsible for SLO violation detection, prediction, and enforcement actions to be performed across all the nodes in the cluster. The enforcement actions will include adding or removing workload replicas (horizontally scaling) and migrating a replica to a resource-richer machine (live migration).

As illustrated above, SLOC will introduce several fundamental improvements to the SLO management in the cloud. At the same time, the design philosophy of our SLOC framework is to adopt best-practices building cloud-native systems and reuse existing stat-of-the-art approaches whenever possible.

RELATED WORK

Depending on explicit considerations of SLOs, users' level of involvement, and support of elasticity concerns we group current approaches into four categories: i) profiling-based, ii) history-based, iii) SLO-based, and iv) black box.

The first category requires a workload to be profiled before it is deployed. PerfIso[13] and Quasar[6] are examples of this approach. PerfIso aims to improve resource utilization on single machines/VMs by scheduling batch jobs next to latency-sensitive (primary) services by dividing the available CPU cores into categories. Profiling of the primary workload is required to determine the size of the buffer category, i.e., cores that can be used by either workload type. Quasar aims to

improve resource utilization in clusters. It requires multiple profiling runs of an application to be able to classify the application and thus determine the efficiency of scaling it vertically and horizontally. Unfortunately, collecting profiling data can be problematic, because in public clouds it is not always possible to obtain this information beforehand.[3] Even if it can be obtained, workloads that were not encountered during profiling can have a negative impact, because the system often does not know how to classify them.[14]

The second category includes approaches that require historical log data as input. Based on load and log information the authors of[21] dynamically instantiate Markov Decision Process models and then use probabilistic model checking to derive elasticity decisions. Resource Central[5] monitors VMs and learns consistent behaviors from these data offline, which makes it also history-based. It then uses these learned behaviors to provide predictions to resource managers. The main drawback of this approach compared with SLOC is that it requires existing log data as input. If this log data is not based on all typical workloads, a similar problem like with profiling-based approaches may occur.

The third category requires the definition of some sort of SLO. The PARTIES[3] resource controller relies on QoS specifications, to schedule multiple latency-critical applications on the same host. Heracles[18] colocates batch jobs with latency-critical applications while maintaining the SLOs of the latency-critical services. Wang *et al.*[23] combine vertical and horizontal scaling to maintain a certain availability (an SLO) of a service, scaling out to achieve the target availability and scaling up to control costs when the SLOs are already fulfilled.

Out of the four categories, this one has the highest similarity with SLOC. It, too, requires developers to specify SLOs for their applications and/or components. However, SLOC will expand the reach of these specifications to include all three elasticity dimensions and will allow developers to specify more high level SLOs than existing approaches, while deriving low level requirements itself.

The fourth category requires no prior knowledge of the applications running on a cluster, but depends solely on the monitoring data and is thus called a *black-box* approach. It may be argued that profiling- and history-based methods are also black-box approaches, because no internal knowledge of the running application needs to be provided by the developer.

However, with a profiling-based method, one or more profiling runs need to be carried out, which serve as input to the resource/elasticity optimizer. History-based approaches require log data to be existent when the optimizer is started. A black-box method needs neither profiling runs nor existing log data. PerfIso requires profiling data, yet its authors call it a *black-box* approach, because it is not concerned about the details of the OS or the primary application. Scavenger[15] aims to improve resource efficiency by scheduling *background* jobs next to *foreground* VMs. To this end, it uses the mean and standard deviation of the foreground tasks' usage of various resources to optimize the contention of memory, network, processor cache, and CPU cores. Combining vertical scaling with horizontal scaling measures results in "significant cost savings over" committing entirely to one of the two techniques.[11] Gandhi *et al.*[11] found scaling up to be typically better when there are strict SLOs, while scaling out being typically superior when there is a high load. However, horizontal scaling needs to take limitations of the architecture of a cloud provider into account as well. ScaleBench[16] is a benchmark for detecting capacity degradation when scaling an application out. Unlike SLOC, black-box approaches do not require any SLOs to be defined for applications. However, these approaches have the limitation that they cannot optimize the resources assigned to a workload according to all the developer's needs. A developer may want a certain response time, but only if it does not exceed a maximum cost. Without specifying what is needed, an optimizer cannot exactly deliver what the developer wants.

CONCLUSION

Cloud service consumption is ever-stronger moving towards API-based consumption models. The interaction with complex business units is done through a well-defined API that shields users from underlying complexities such as microservice deployment topologies and service meshes. In spite of the practical importance SLOs and

SLAs have for consuming contemporary cloud services, vast majority of cloud platforms provide support for SLAs only in terms of statically predefined SLOs, e.g., service availability, and low-level resource capacity guarantees, e.g., CPU usage. Furthermore, to date there is only limited support to clearly map workload performance requirements to the resource capacity guarantees. In this article, we introduced the SLOC framework. The core idea behind our SLOC framework is to enable SLO-native management of elastic Cloud resources. We discussed how our SLO-native approach introduces a paradigm shift from general, business logic agnostic, low-level SLAs to intent-based, SLO-first, performance-driven, and orchestration-aware elasticity models.

One of SLOC's main objectives is to exploit and advance current support for managing elasticity concerns in order to achieve better support for SLOs.

To this end, we intend to build on our previous work on elasticity space and boundaries as well as to facilitate defining and enforcing soft and hard SLO constraints in terms of elasticity mechanisms. The core of SLOC's approach revolves around three main concepts: SLO elasticity policy language, SLO observability, and SLO elasticity controlling. We provided an overview how these concepts symbiotically enable the paradigm shift toward SLO-native cloud computing, hence unlocking the potential of coherent, congruent, full-stack elasticity strategies to efficiently deal with workload SLOs and SLAs. In the future, we will continue our work along the presented road map in order to deliver an open source implementation of our SLOC framework.

ACKNOWLEDGMENTS

This work was supported by Futurewei's Cloud Laboratory as part of the overall open source initiative.

REFERENCES

1. Kubernetes. 2020. [Online]. Available: https://kubernetes.io/, Accessed on: Apr. 2020.
2. R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 24, no. 30, pp. 599–616, 2009.
3. S. Chen, C. Delimitrou, and J. F. Martínez, "Parties: QoS-aware resource partitioning for multiple interactive services," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 107–120.
4. G. Copil, D. Moldovan, H. Truong, and S. Dustdar, "SYBL: An extensible language for controlling elasticity in cloud applications," in *Proc. 13th IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput.*, May 2013, pp. 112–119.
5. E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proc. 26th Symp. Oper. Syst. Princ.*, 2017, pp. 153–167.
6. C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," *ACM Special Interest Group Program. Lang.*, vol. 49, no. 4. pp. 127–144, Feb. 2014.
7. S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong, "Principles of elastic processes," *IEEE Internet Comput.*, vol. 15, no. 5, pp. 66–71, Sep./Oct. 2011.
8. R. Engle *et al.* "Capacity reclamation and resource adjustment," U.S. Patent 9,038,068, May 19, 2015.
9. Futurewei Technologies, Inc., "Arktos open source project repository," 2020. [Online]. Available: https://github.com/futurewei-cloud/arktos, Accessed on: Apr. 2020.
10. Futurewei Technologies, Inc., "Mizar virtual network data plane open source project," 2020. [Online]. Available: https://github.com/futurewei-cloud/mizar, Accessed on: Apr. 2020.
11. A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Modeling the impact of workload on cloud resource scaling," in *Proc. IEEE 26th Int. Symp. Comput. Archit. High Perform. Comput.*, Oct. 2014, pp. 310–317.
12. R. Householder, S. Arnold, and R. Green, "On cloud-based oversubscription," 2014, *Int. J. Eng. Trends and Tech. (IJETT)*, vol. 8, no. 8, pp. 425–431, 2014.
13. C. Iorgulescu *et al.*, "Perfi performance isolation for commercial latency-sensitive services," in *Proc. USENIX Annu. Tech. Conf.*, Jul. 2018, pp. 519–532.
14. S. A. Javadi and A. Gandhi, "Dial: Reducing tail latencies for cloud applications via dynamic interference-aware load balancing," in *Proc. IEEE Int. Conf. Auton. Comput.*, Jul. 2017, pp. 135–144.
15. S. A. Javadi, A. Suresh, M. Wajahat, and A. Gandhi, "Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 272–285.

16. Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "The limit of horizontal scaling in public clouds," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 5, no. 1, Feb. 2020, Art. no. 6.
17. A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," *J. Netw. Syst. Manage.*, vol. 11, no. 1, pp. 57–81, 2003.
18. D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Improving resource efficiency at scale with Heracles," *ACM Trans. Comput. Syst.*, vol. 34, no. 2, May 2016, Art. no. 6.
19. D. Moldovan, G. Copil, H. Truong, and S. Dustdar, "On analyzing elasticity relationships of cloud services," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, 2014, pp. 447–454.
20. D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar, "MELA: Elasticity analytics for cloud services," *Int. J. Big Data Intell.*, vol. 2, no. 1, pp. 45–62, 2015.
21. A. Naskos *et al.*, "Dependable horizontal scaling based on probabilistic model checking," in *Proc. 15th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2015, pp. 31–40.
22. S. M. Noonan, "Managing resource bursting," U.S. Patent 9 417 902, Aug. 16, 2016.
23. W. Wang, H. Chen, and X. Chen, "An availability-aware virtual machine placement approach for dynamic scaling of cloud applications," in *9th Int. Conf. Ubiquitous Intell. Comput. 9th Int. Conf. Auton. Trusted Comput.*, Sep. 2012, pp. 509–516.

Stefan Nastic is co-founder and CEO of Reinvent Labs. He has a track-record as a cloud solution architect, consultant and cloud-native engineer working on various commercial and research projects for over a decade. He received his Dr. Tech. degree (equivalent PhD) from TU Wien in 2016. He is the corresponding author of this article. Contact him at snastic@reinvent-group.at.

Andrea Morichetta is currently a University Assistant with the Distributed Systems Group, TU Wien, Vienna, Austria. His research focuses on data analysis and machine learning, network monitoring, and cloud computing. He received the Ph.D. degree with the Telecommunication Network Group, Politecnico di Torino, Turin, Italy, January 2020. Contact him at a.morichetta@dsg.tuwien.ac.at.

Thomas Pusztai is currently working toward the Ph.D. degree with the Distributed Systems Group, TU Wien, Vienna, Austria. He worked for five years in industry, the final two as Senior Developer and Team Lead. His interests include software engineering, model-driven engineering, and cloud computing. Contact him at t.pusztai@dsg.tuwien.ac.at.

Schahram Dustdar is currently a Full Professor of computer science heading the Distributed Systems Group, TU Wien, Vienna, Austria. His work focuses on Internet technologies. He is an IEEE Fellow, a member of the Academia Europaea—The Academy of Europe, and an ACM Distinguished Scientist. Contact him at dustdar@dsg.tuwien.ac.at.

Xiaoning Ding is currently a Senior Director and Architect with Cloud Lab, Futurewei Technologies, Santa Clara, CA, USA. His research interests include cloud infrastructure, machine learning infrastructure, and distributed system. He received the Ph.D. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2007. Contact him at xiaoning.ding@futurewei.com.

Deepak Vij is currently a Seasoned Technologist with Futurewei Technologies, Santa Clara, CA, USA, and formerly Chair of IEEE P2302 cloud computing initiative. He has an extensive technology leadership background as technology visionary, board member, and evangelist. His extensive technology background includes areas such as cloud computing, distributed computing, security, business intelligence, etc. Contact him at deepak.vij@futurewei.com.

Ying Xiong is currently the Head of Cloud Lab, and Technical VP with Futurewei Technologies, Santa Clara, CA, USA. His research interests include cloud and edge computing, virtual network, and large scale network management in the cloud, as well as AI platform for optimizing ML applications. He received the Ph.D. degree in computer and information system. He has more than 20 years of experience in distributed system design and large scale cloud platform implementation. Contact him at ying.xiong@futurewei.com.