A Quantitative Approach to Coordinated Scaling of Resources in Complex Cloud Computing Workflows

Laura Carnevali[1], Marco Paolieri[2], Benedetta Picano[1], Riccardo Reali[1](✉), Leonardo Scommegna[1], and Enrico Vicario[1]

[1] Department of Information Engineering, University of Florence, Florence, Italy
{laura.carnevali,benedetta.picano,riccardo.reali, leonardo.scommegna,enrico.vicario}@unifi.it

[2] Department of Computer Science, University of Southern California, Los Angeles, USA
paolieri@usc.edu

Abstract. Resource scaling is widely employed in cloud computing to adapt system operation to internal (i.e., application) and external (i.e., environment) changes. We present a quantitative approach for coordinated vertical scaling of resources in cloud computing workflows, aimed at satisfying an agreed Service Level Objective (SLO) by improving the workflow end-to-end (e2e) response time distribution. Workflows consist of IaaS services running on dedicated clusters, statically reserved before execution. Services are composed through sequence, choice/merge, and balanced split/join blocks, and have generally distributed (i.e., non-Markovian) durations possibly over bounded supports, facilitating fitting of analytical distributions from observed data. Resource allocation is performed through an efficient heuristics guided by the mean makespans of sub-workflows. The heuristics performs a top-down visit of the hierarchy of services, and it exploits an efficient compositional method to derive the response time distribution and the mean makespan of each sub-workflow. Experimental results on a workflow with high concurrency degree appear promising for feasibility and effectiveness of the approach.

Keywords: Cloud computing · coordinated scaling · stochastic workflow · end-to-end response time distribution · complex workflow structure

1 Introduction

Cloud Computing (CC) systems [7,16] need to store, manage, and process enormous amounts of data continuously generated by a variety of sources within the Internet of Things (IoT) [28]. Excessive network traffic or heavy computational workload may lead to violations of Quality of Service (QoS) attributes granted through Service level Agreements (SLAs) [29]. Therefore, CC systems must autonomously adapt their operation in response to time-varying changes both in the software system itself and in its operating environment [26,32]. Adaptation can be achieved through *autoscaling* systems [13], which dynamically change software configurations and provision hardware resources on demand, with the goal of continuously satisfying cost objectives as well as non-functional Service Level Objectives (SLOs), i.e., specific measures agreed within an SLA. Scaling actions can be *horizontal*, if the system adds or removes containers or Virtual Machines (VMs) where services can be deployed, or *vertical*, if the system changes specifications of those containers or VMs, e.g., CPU cores or available memory.

Horizontal scaling can optimize resource provisioning for *individual* services orchestrated in larger applications [1,3,14,20,37], e.g., composite web services [8], Functions as a Service (FaaS) platforms [23,33], microservice architectures [2]. In [20], bottlenecks in a multi-tier application are automatically detected and resolved, minimizing the number of web servers and database instances while guaranteeing a maximum response time. Dynamic scaling of the number of VMs in cloud services is performed based on the number of active sessions of each web server instance [14], using queueing theory to estimate demand [1], and leveraging also time series analysis to forecast load intensity [3]. Few approaches exploit *coordinated* scaling of resources to avoid undesired effects of *local* scaling like bottleneck shifting and oscillations [37], e.g., by reconfiguring services of small web applications together [35], by exploiting time-series analysis and queueing theory to determine the number of VM instances that minimizes energy consumption without violating SLAs [4], or by collectively providing application tiers with a number of servers or VMs that guarantees meeting contracted [37] or average response times [6]. Though horizontal scaling has received more attention [13] and has better support from cloud vendors [15], being easier to implement and manage, it performs coarse-grained adaptation through static replication of VMs or containers with fixed-size configurations, and it suffers from non-negligible lags to instance and start VMs or containers [39], which, despite lag-mitigating actions like dynamic VM cloning [24], may negatively affect time-critical applications.

Vertical scaling performs fine-grained resource adaptation by modifying attributes of VMs or containers [15,21,34], thus limiting resource over-provisioning and resulting preferable for applications with time-critical requirements. In [34], CPU voltage and frequency of VMs in multi-tenant cloud systems are individually adapted to meet SLOs, supporting migration to new VMs in case of overloading. Optimization of CPU usage and memory allocated to a cloud application is performed in [15] to meet requirements on mean response time, exploiting a performance model based on an inverse relationship between the application mean response time and the number of allocated CPU cores [21]. In [39], CPU power tuning and hotplugging are performed to improve CPU usage efficiency in a web server, with minimum SLA violation rate. Few approaches perform vertical scaling in a *coordinated* manner. In [22], soft resources of web application servers (e.g., number of server threads and database connections) are allocated based on measured throughput and concurrency. A resource-management framework is defined in [25] to manage shared resources among microservices, exploiting machine learning methods both to localize microservice instances responsible for SLO violations and to define methods to mitigate resource contention. Horizontal and vertical scaling are combined in [19] to determine a load distribution policy for co-located distributed applications by exploiting multi-class queueing networks and model predictive control, and in [17] to adapt the number of replicas and the CPU capacity of each microservice by using layered queueing networks to assess potential performance improvement of scaling actions.

The few approaches that address coordinated resource scaling [26] mainly consider simple cloud applications consisting of few services orchestrated as sequential workflows. Notably, no approach takes into account the end-to-end (e2e) response time distribution in scaling decisions, which instead becomes relevant when SLAs are characterized by soft deadlines and penalty functions [27] defined as rewards calculated from such distribution.

In this paper, we present an efficient approach to perform *coordinated* vertical scaling of resources in complex stochastic workflows, aimed at satisfying a SLO by improving the workflow *e2e response time distribution*. Specifically, workflows compose IaaS services running on dedicated clusters whose size must be determined in advance, reserving and statically assigning resources to services before execution. Services are composed through sequence, fork-join, and choice-merge patterns [30], and have generally distributed (GEN) response times possibly with bounded supports, facilitating representation of real-time constraints and fitting of analytical distributions from observed data. Each service is characterized by a job size [5], representing its makespan (i.e. its expected response time) with a given amount of assigned resources; we assume the makespan to be inversely proportional to the amount of assigned resources [15,21,31]. The defined heuristics uses a structured workflow model [10] to perform a top-down visit of the hierarchy of services, assigning resources so as to minimize the makespan of the workflow e2e response time and to satisfy the agreed SLO. To this end, the heuristics exploits an efficient compositional analysis method [11] to derive the response time distribution of each sub-workflow and to compute its makespan. Feasibility and effectiveness of the approach are assessed on a non-trivial synthetic workflow stressing computational complexity. Results show that the heuristics is effective at improving the e2e response time distribution of the entire workflow, and very efficient, enabling its application at runtime in reaction to QoS changes.

In the framework of [9], our approach is defined by the following attributes: the *goal of resource adaptation* is to ensure that workflow execution fulfils non-functional requirements specified by percentiles of quality attributes, i.e., that the makespan of the workflow response time satisfies the agreed SLO; the *stage of system lifetime* at which resource adaptation is performed is the runtime stage with proactive mode (i.e., anticipating resource adaptation), though the approach can be applied also in reactive mode (i.e., after changes in quality attributes) as shown by the experimental results; the *composition level* at which resource adaptation is performed involves both services (i.e., abstract composition made of tasks orchestrated by some composition logic) and workflow (i.e., concrete composition where tasks of an abstract composition are mapped to implementations); the *scope of resource adaptation*, in terms of *number of systems* and *granularity* of adaptation, considers a single system and a single request; *adaptation actions* mainly consist of service tuning operations changing behavior of concrete services (e.g., reducing the makespan by increasing the amount of resources), though adaptation is performed in a coordinated manner; and, resource adaptation is performed by a *single authority*.

The rest of the paper is organized as follows. Section 2 recalls the hierarchical formalism for workflow modeling and the compositional method for evaluation of the workflow e2e response time distribution. Section 3 illustrates the proposed resource assignment method. Section 4 presents the experimental results achieved on a complex workflow. Finally, Sect. 5 draws conclusions and outlines possible extensions and improvements of the proposed approach.

2 Background: Workflow Modeling and Evaluation

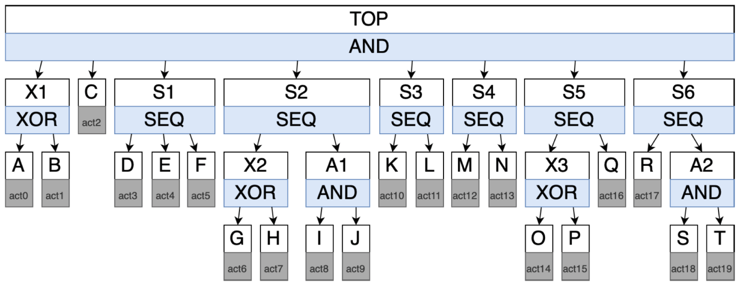
We model workflows as recursive compositions of blocks specified by Stochastic Time Petri Nets (STPNs) [38]. Each STPN block has a single starting place, which receives a token when workflow execution starts, and a single final place, which receives a token when workflow execution eventually ends with probability 1 (w.p.1). As shown in Fig. 1, blocks model sequential, balanced split/join, and choice/merge workflow patterns [30,40], with the following EBNF syntax:

$$\begin{aligned}\textsc{Block} := \mathrm{ACT} \mid \mathrm{SEQ}\{\textsc{Block}_1, \ldots, \textsc{Block}_n\} \\ \mid \mathrm{AND}\{\textsc{Block}_1, \ldots, \textsc{Block}_n\} \mid \mathrm{XOR}\{\textsc{Block}_1, \ldots, \textsc{Block}_n, p_1, \ldots, p_n\}\end{aligned} \tag{1}$$

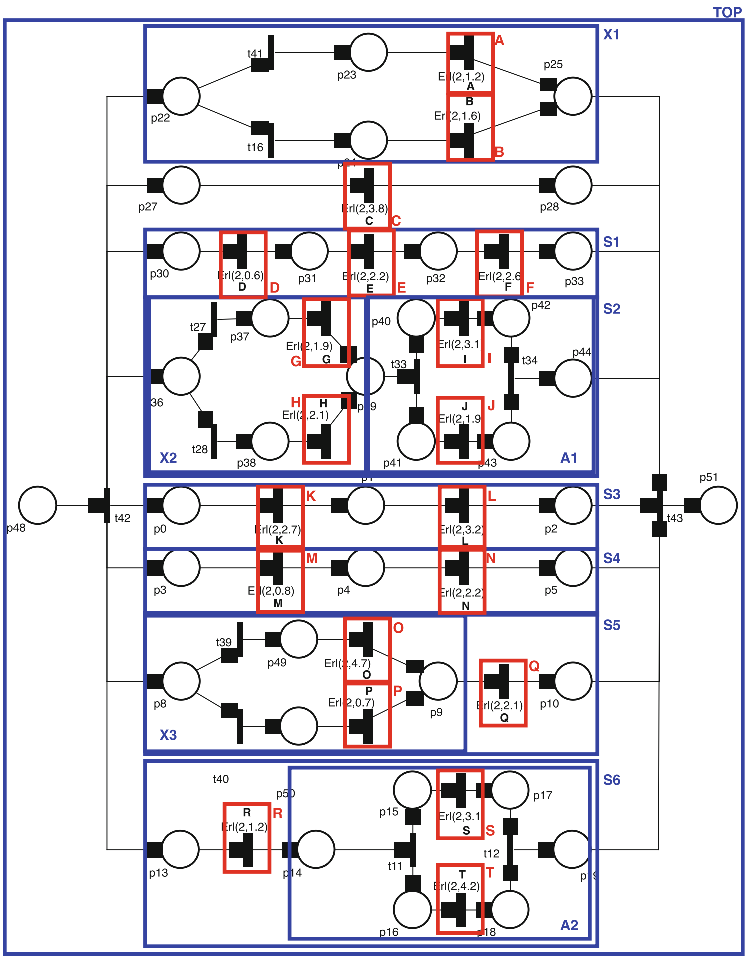
where ACT is an elementary activity with non-Markovian distribution possibly with bounded support (e.g., block A in Fig. 1b), $\mathrm{SEQ}\{\textsc{Block}_1, \ldots, \textsc{Block}_n\}$ models n sequential blocks $\textsc{Block}_1, \ldots, \textsc{Block}_n$ (e.g., block S1 in Fig. 1b), $\mathrm{AND}\{\textsc{Block}_1, \ldots, \textsc{Block}_n\}$ models n concurrent blocks $\textsc{Block}_1, \ldots, \textsc{Block}_n$ (e.g., block A1 in Fig. 1b), and $\mathrm{XOR}\{\textsc{Block}_1, \ldots, \textsc{Block}_n, p_1, \ldots, p_n\}$ models n alternative blocks $\textsc{Block}_1, \ldots, \textsc{Block}_n$ with probability $p_1, \ldots, p_n$, respectively (e.g., block X1 in Fig. 1b). Note that associating activity durations with non-Markovian distributions possibly with bounded supports facilitates fitting of analytical distributions from data collected from real web applications and enables representation of firm constraints on execution times of activities.

This workflow model can be represented as a *structure tree* [10] $S = \langle N, n_0 \rangle$, where N is the set of nodes (i.e., blocks) and $n_0 \in N$ is the root node (i.e., the entire workflow). In turn, each node $n_i \in N$ is a tuple $\langle C_i, \mathrm{type}_i \rangle$, where C_i is the set of the children nodes of n_i and $\mathrm{type}_i \in \{\mathrm{ACT}, \mathrm{SEQ}, \mathrm{AND}, \mathrm{XOR}\}$ is the type of the block modeled by node n_i, e.g., in Fig. 1a, node A1 models an AND block composing nodes I and J.

For complex workflows made of several concurrent activities with duration characterized by GEN Cumulative Distribution Functions (CDFs) possibly with bounded supports, the e2e response time distribution cannot be evaluated by



(a)



(b)

Fig. 1. (a) Structure tree and (b) STPN modeling a synthetic workflow.

transient analysis [18] of the workflow STPN. To address the issue, a compositional approach is defined in [11], which first performs a top-down visit of the structure tree to estimate the analysis complexity of blocks, then evaluates the response time distribution of the identified sub-workflows in isolation, and finally performs a bottom-up recomposition of the obtained results. In particular, for workflows defined by well-nested composite blocks as in this paper (i.e., composition of AND, SEQ, and XOR blocks), the exact e2e response time distribution can be evaluated by recursive numerical analysis [11].

3 Coordinated Resource Scaling Heuristics

In this section, we present a vertical scaling heuristics for coordinated allocation of resources to each workflow activity, improving the workflow e2e response time distribution. We illustrate how the workflow structure tree is visited (Sect. 3.1) and the scaling decisions for each node type (Sect. 3.2).

3.1 Heuristics Overview

The response time of each activity is a random variable depending on the amount R of allocated resources. The *job size* X of each activity is the invariant amount of work to be completed with the assigned amount R of resources, and it is evaluated as a scalar value depending on R and on a mean makespan T_R, i.e., the mean response time. Similarly to [15,21,31], we consider $X := R\,T_R \;\forall\, R$. For each node of the structure tree of a workflow, given a new resource allocation, the makespan can be estimated as a function of the invariant job sizes of the children of the node. This function depends on the node type.

The heuristics performs a top-down visit of the structure tree and splits the resources assigned to each node among its children by solving an optimization problem, until the resource allocation of all elementary activities is determined. Note that, as long as the job sizes of the tasks are known, the method can identify an ideal resource allocation not only when the total amount of resources is redistributed, but also when it is incremented or decremented. Hence, different scalar SLOs can be fit by varying the amount of resources in input.

Let $S = \langle N, n_0\rangle$ be a structure tree as defined in Sect. 2. We extend the definition of node $n_i \in N$ with the tuple $\langle C_i, \text{type}_i, R_i, T_i, X_i\rangle$, where $R_i \in \mathbb{R}_{>0}$ is the amount of resources initially assigned to n_i, $T_i \in \mathbb{R}_{\geq 0}$ is the makespan of n_i, and X_i is the job size of n_i. For each non-leaf node $n_i \in N$, the number of resources of n_i is the sum of the number of resources allocated to its children nodes, i.e., $\forall\, n_i \in N$ such that $C_i \neq \emptyset$, $R_i = \sum_{n_j \in C_i} R_j$. To coordinately adapt the resource provisioning of the activities of a workflow, the approach performs a *top-down* visit of the workflow structure tree.

- Initially, an arbitrary amount of resources R_0^{in} is assigned to the root node n_0.
- For each non-leaf node n_k, the amount of input resources R_k^{in} is split by assigning an amount $R_j^\star$ to each child node n_j, i.e., $\sum_{n_j \in C_k} R_j^\star = R_k^{\text{in}}$; the assignment depends on the node type.

By induction, the sum of the amounts of resources allocated to the leaf nodes is equal to the amount of resources of the root node, i.e., $\sum_{n_k | C_k = \emptyset} R_k^\star = R_0^{\text{in}}$.

3.2 Scaling Decisions

We characterize the different resource scaling decisions based on the node types.

Sequential Activities. Let n_k be an activity with type$_k$ = SEQ and children $C_k = \{i, j\}$, and let R_k^{in} be the amount of resources to be split. The makespan of node n_k can be obtained as

$$T_k = T_i + T_j = \frac{X_i}{R_i} + \frac{X_j}{R_k^{\text{in}} - R_i} \tag{2}$$

which has a minimum when the following resources are allocated to node n_i:

$$R_i^\star = \frac{\sqrt{X_i}}{\sqrt{X_i} + \sqrt{X_j}} R_k^{\text{in}}\,. \tag{3}$$

The result is obtained by imposing $\frac{dT_k}{dR_i} = 0$, and it can be extended by induction to the ordered sequence $SEQ(n_1, \ldots, n_J)$ of $J > 2$ activities:

$$R_i^\star = \frac{\sqrt{X_i}}{\sum_{j=1}^{J} \sqrt{X_j}} R_k^{\text{in}} \quad \forall\, i \in \{1, \ldots, J\}. \tag{4}$$

Each node allocation is thus obtained considering the allocation evaluated for previous nodes, that is removed from the input resources R_k^{in}. Note that, since we reserve resources before service execution, the optimum does not exploit resources used by activities that have already been executed.

Concurrent Activities. Let n_k be an activity with type$_k$ = AND, children $C_k = \{i, j\}$, and amount R_k^{in} of resources to be split. In this case, the makespan (mean response time) cannot be defined as an analytical function of R_i, precluding to evaluate the minimum by exploiting the Fermat theorem. Hence, we provide a heuristics evaluation of $R_i^\star$ depending on a parameter $\alpha \in \mathbb{R}^+$ modulating the weight of the job size in determining the solution. In particular, $R_i^\star$ is evaluated by imposing equality between the ratio of response times of node n_k children, and the $(\alpha + 1)$-th power of the resources provisioned to the children:

$$\frac{X_i}{R_i^{\alpha+1}} = \frac{X_j}{(R_k^{\text{in}} - R_i)^{\alpha+1}} \tag{5}$$

This leads to the allocation:

$$R_i^\star = \frac{\sqrt[\alpha+1]{X_i}}{\sqrt[\alpha+1]{X_i} + \sqrt[\alpha+1]{X_j}} R_k^{\text{in}} \tag{6}$$

The solution is extended to J activities by induction:

$$R_i^\star = \frac{\sqrt[\alpha+1]{X_i}}{\sum_{j=1}^{J} \sqrt[\alpha+1]{X_j}} R_k^{\text{in}} \quad \forall\, i \in \{1, \ldots, J\} \tag{7}$$

Note that, when $\alpha = 0$, the heuristics determines $R_i^\star$ by imposing equality between the response times of the considered node children.

Alternative Activities. Let n_k be an activity with $type_k = \text{XOR}$, children $C_k = \{i, j\}$ having probabilities p_i and $p_j = 1 - p_i$ to occur, and R_k^{in} the amount of resources to be split. The makespan of n_k is

$$T_k = p_i T_i + p_j T_j = p_i \frac{X_i}{R_i} + p_j \frac{X_j}{R_k^{\text{in}} - R_i} \tag{8}$$

which has the minimum

$$T_k^\star = \frac{\left(\sqrt{p_i X_i} + \sqrt{p_j X_j}\right)^2}{R_k^{\text{in}}} \tag{9}$$

when the following resources are allocated to node i:

$$R_i^\star = \frac{\sqrt{p_i X_i}}{\sqrt{p_i X_i} + \sqrt{p_j X_j}} R_k^{\text{in}} \tag{10}$$

which is in turn obtained by exploiting the Fermat theorem. As the solution shows, the optimal allocation of an XOR node is a generalization of the optimal allocation for a SEQ node. Hence, the extension to the case with more than 2 activities can be derived from the solution obtained for the SEQ node. Also note that the available resources R_k^{in} are split among the activities i and j: this assumption is useful to model workflows in microservice and service-oriented applications, where each service has a reserved amount of resources. In contrast, for microservices deployed with FaaS cloud solutions, resources are allocated on-demand for each service execution: in this case, cloud costs are accrued only for the resources of the selected service; the expected cost is in this case $p_i R_i + p_j R_j$ instead of $R_i + R_j$, resulting in a different optimal allocation.

Once resources are assigned to each simple node, the response time CDF of each activity can be determined by leveraging linearity of the job size invariance, e.g., if resources assigned to an activity with Erlang response time distribution are doubled, then the rate of the Erlang distribution doubles.

4 Experimental Evaluation

In this section, we assess feasibility and effectiveness of the proposed heuristics. First, we show how different values of parameter α produce different resource allocations, with consequent different improvement of the workflow e2e response time CDF (Sect. 4.1). The experiment is performed for two different initial resource allocations. Then, we test the ability of the heuristics to meet an agreed SLO while minimizing the amount of allocated resources (Sect. 4.2).

Both experiments are performed on the synthetic workflow of Fig. 1, which consists of 20 elementary blocks combined through well-nested patterns, yielding a model with up to 10 concurrent activities. For each elementary activity,

Table 1. Resources allocated to the activities of the workflow of Fig. 1, before (column r_{start}) and after heuristics execution with different values of α (columns r_α with $\alpha \in \{0, \frac{1}{4}, \frac{1}{2}, 1, 2, 4\}$): (a) balanced and (b) unbalanced initial allocation.

ACT	r_{start}	r_0	$r_{1/4}$	$r_{1/2}$	r_1	r_2	r_4
A	1.00	0.52	0.64	0.73	0.85	0.97	1.07
B	1.00	0.55	0.68	0.78	0.90	1.03	1.14
C	1.00	0.25	0.41	0.57	0.84	1.23	1.65
D	1.00	2.85	2.51	2.29	2.01	1.75	1.54
E	1.00	1.49	1.31	1.19	1.05	0.91	0.81
F	1.00	1.37	1.20	1.01	0.97	0.84	0.74
G	1.00	0.72	0.71	0.70	0.68	0.65	0.62
H	1.00	0.84	0.83	0.82	0.79	0.75	0.72
I	1.00	0.63	0.66	0.68	0.69	0.69	0.68
J	1.00	1.03	0.98	0.94	0.88	0.81	0.75
K	1.00	0.54	0.67	0.77	0.89	1.03	1.15
L	1.00	0.50	0.61	0.70	0.82	0.95	1.05
M	1.00	1.61	1.67	1.69	1.69	1.68	1.65
N	1.00	0.97	1.00	1.02	1.02	1.01	0.99
O	1.00	0.48	0.47	0.46	0.44	0.42	0.40
P	1.00	1.70	1.66	1.63	1.56	1.48	1.40
Q	1.00	1.19	1.16	1.14	1.09	1.04	0.98
R	1.00	1.49	1.52	1.52	1.52	1.49	1.45
S	1.00	0.71	0.71	0.70	0.68	0.66	0.63
T	1.00	0.53	0.56	0.57	0.59	0.59	0.59

(a) Balanced initial res. allocation.

ACT	r_{start}	r_0	$r_{1/4}$	$r_{1/2}$	r_1	r_2	r_4
A	0.375	0.21	0.32	0.43	0.60	0.80	0.99
B	0.375	0.22	0.34	0.45	0.63	0.85	1.05
C	0.375	0.01	0.21	0.33	0.59	1.01	1.52
D	0.375	1.13	1.27	1.34	1.41	1.44	1.42
E	0.375	0.59	0.66	0.70	0.74	0.75	0.74
F	0.375	0.54	0.61	0.65	0.68	0.69	0.68
G	3.75	2.49	2.22	2.02	1.76	1.48	1.26
H	0.375	0.92	0.82	0.75	0.65	0.55	0.46
I	3.75	3.14	2.63	2.28	1.83	1.40	1.08
J	0.375	0.51	0.62	0.68	0.74	0.77	0.75
K	0.375	0.21	0.34	0.45	0.63	0.85	1.05
L	0.375	0.20	0.31	0.41	0.58	0.78	0.97
M	0.375	0.64	0.84	0.99	1.19	1.38	1.51
N	0.375	0.39	0.51	0.60	0.72	0.83	0.91
O	3.75	2.05	1.93	1.83	1.67	1.47	1.29
P	0.375	2.29	2.16	2.04	1.86	1.64	1.44
Q	0.375	1.01	0.95	0.90	0.82	0.72	0.63
R	0.375	1.46	1.44	1.40	1.32	1.20	1.09
S	3.75	2.68	2.51	2.33	2.04	1.67	1.34
T	0.375	0.20	0.31	0.41	0.55	0.70	0.80

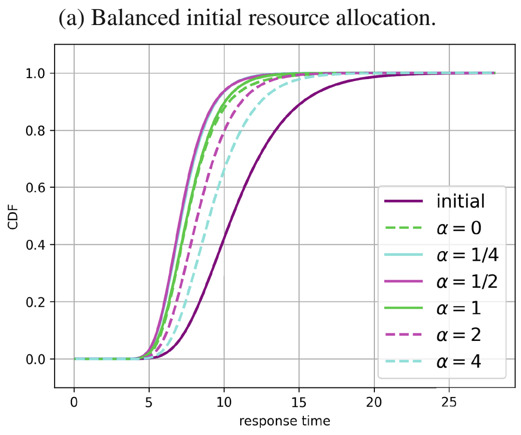
(b) Unbalanced initial res. allocation.

we assume that the response time CDF achieved with a given resource assignment is known, either from measurements of previous implementations or by contract. To easily manage the linear relation between response time and allocated resources assumed by the performance model of Sect. 3.1, and to facilitate the interpretation of the experimental results, without loss of generality, we consider Erlang CDFs for the response times of elementary activities. In particular, we consider Erlang CDFs with 5-phases and rates randomly selected in $[0, 5]$, so guaranteeing variability in expected response times of activities. We remark that any numerical CDF could be considered as well, or any analytical CDF in the class of expolynomial functions (also termed exponomials [36]) supported by the compositional analysis technique of [11] exploited by the proposed heuristics.

All the experiments reported in this section have been performed on a MacBook Pro 2021 equipped with an Apple M1 Pro octa-core processor with a rate clock up to 3.2 GHz and 16 GB RAM. Experiments have been performed using an extension of the Eulero Java library [12], currently under development.

4.1 Heuristics Sensitivity to Parameter α

We consider two arbitrary different initial allocations of resources to the activities of the workflow of Fig. 1. In the first (column r_{start} of Table 1a), we consider a



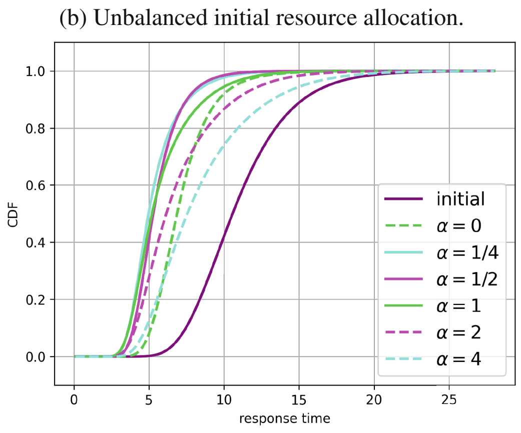


Fig. 2. CDF of the e2e response time of the workflow of Fig. 1, obtained through the execution of the proposed heuristics with different values of parameter α, by assuming: (a) the balanced initial resource allocation shown in Table 1a, and (b) the unbalanced initial resource allocation shown in Table 1b.

balanced allocation where each activity is assigned 1 resource, for a total amount of 20 resources (*balanced initial resource allocation*). In the second (shown in column r_{start} of Table 1b), we consider an unbalanced allocation, where activities G, I, O, and S have 10 times the resources of the other activities (*unbalanced initial resource allocation*). To make results comparable, we maintain a total of 20 resources, thus allocating 15 resources to G, I, O, and S (i.e., 3.75 resources each), and 5 resources to the remaining activities (i.e., 0.375 each). Tables 1a and 1b also show the resource allocation computed by the proposed heuristics.

We evaluate how the resource allocation provided by the heuristics improves the workflow e2e response time CDF for different values of $\alpha \in \{0, \frac{1}{4}, \frac{1}{2}, 1, 2, 4\}$. Results reported in Figs. 2a and 2b show an improvement for any considered value of α and for any of the two initial allocations of resources, with better results achieved for $\alpha \in [0, 1]$ and nearly the best result obtained for $\alpha = \frac{1}{4}$. This result suggest that, being $\alpha = \frac{1}{n}$, the e2e response time CDF improves as n increases, up to a certain value beyond which the CDF gets worse.

If the initial resource allocation is balanced, different values of $\alpha \in [0, 1]$ produce nearly comparable e2e response time CDFs. This result suggests that, in this case, balancing the makespan of the children of AND nodes determines a good resource allocation, significantly improving the workflow e2e response time CDF. Conversely, if the initial resource allocation is unbalanced, better results are obtained for values of $\alpha \in (0, 1]$, with a worse result obtained for $\alpha = 0$ with respect to the case of balanced initial allocation of resources.

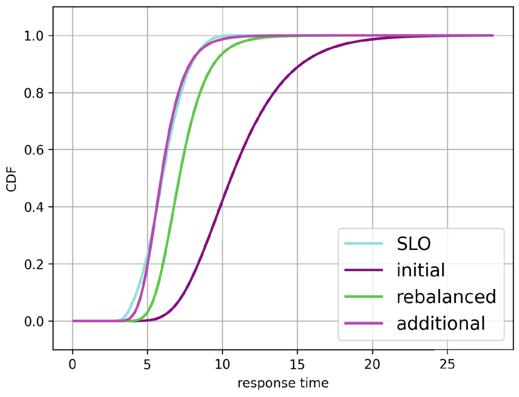
It is worth noting that, for both variants of the experiment, the proposed heuristics runs in 0.2 s on average, proving to be efficient even for complex workflows, which is an essential requirement for modern microservice architectures where hundreds of services are orchestrated as workflows of activities.

4.2 Heuristics Ability to Achieve SLO Guarantees

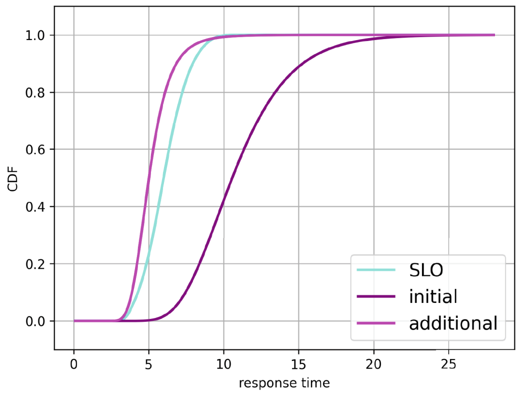
For the workflow of Fig. 1 with the initial balanced resource allocation of Table 1a, we consider an SLO expressed in terms of e2e response time CDF. In particular, the SLO is obtained as the e2e response time CDF of a randomly generated well-nested workflow, having expected response time equal to $T_{\text{SLO}} = 6.17$ s. As shown in Fig. 3, with this allocation of resources, the workflow e2e response time CDF (violet curve) does not meet the SLO (light blue curve). Rebalancing the existing resources by applying the proposed heuristics with $\alpha = \frac{1}{4}$ (which is the value yielding the best results in Sect. 4.1) produces an improvement of the e2e response time CDF, which however still violates the required SLO. Therefore, an additional amount of resources is needed not to violate the SLO.

To determine the new amount of resources, we consider two different strategies whose results are reported in Fig. 3. We perform an experiment (*reallocate resources first*) where we first compute a new resource allocation through our heuristics with $\alpha = \frac{1}{4}$ (green curve in Fig. 3a), by assuming that the total amount of allocated resources does not change, i.e., $R_0 = 20$. Then, we evaluate the additional amount of resources that is needed to meet the specified SLO, by exploiting the assumption of invariance of the job size of a workflow (as discussed in Sect. 3.1). In fact, by knowing the expected response time T_{SLO} of the SLO, the initial amount of allocated resources $R_0 = 20$, and the expected response time T_0 obtained after resource allocation (i.e., considering the resource allocation of column $r_{1/4}$ of Table 1a as initial resource allocation), the amount of resources required not to violate the SLO can be computed as $R^\star = \frac{R_0 T_0}{T_{\text{SLO}}}$. In particular, the additional amount of resources turns out to be equal to 4.06. Finally, the amount $R^\star$ is allocated to the activities using the heuristics, and a new e2e response time CDF is computed (fuchsia curve in Fig. 3a).

Then, we perform a variant of the experiment (*add resources first*), where we directly evaluate the additional amount of resources needed to meet the specified SLO as $R^{\star\prime} = \frac{R_0 T_0'}{T_{\text{SLO}}}$, where T_0' is the workflow expected response time obtained



(a) Reallocate resources first.



(b) Add resources first.

Fig. 3. CDF of the e2e response time of the workflow of Fig. 1, obtained through two strategies: (a) reallocating resourced through the heuristics, determining the amount of resourced needed to satisfy the SLO, and allocating resources again through the heuristics; and, (b) determining the amount of resourced needed to satisfy the SLO and allocating resources through the heuristics.

by considering the initial resource allocation of column r_{start} of Table 1a. In particular, the additional amount of resources turns out to be equal to 16.15. The allocation of the increased amount of resources through our heuristics yields a new e2e response time CDF (fuchsia curve in Fig. 3b).

Figure 3 shows that the e2e response time CDF provided by the reallocate-resources-first strategy is stochastically larger than the one provided by the add-resources-first strategy, and is characterized by a larger expected response time. However, the add-resources-first strategy allocates a larger number of resources, which actually turn out to be over-provisioned, given that the obtained e2e response time CDF is stochastically lower than the SLO. Moreover, note that the time to calculate the new resource provisioning is 0.89 s for the reallocate-resources-first strategy and 0.51 s for the add-resources-first strategy, meaning that there is not a significant loss in performance when the heuristics is executed twice. Therefore, the reallocate-resources-first strategy is preferable.

5 Conclusions

We have presented a heuristics to perform coordinated scaling of resources in cloud computing workflows, with the aim of improving the e2e response time CDF. The heuristic is developed around the concept of job size of an activity, which is assumed to be invariant with respect to the amount of resources provisioned to the activity, and it is guided by the mean makespan indicators of sub-workflows. The method has been successfully tested on a complex workflow with a high degree of concurrency, and applied to the problem of identifying additional resources needed to guarantee a given SLO, proving to be not only effective at improving the workflow e2e response time CDF but also efficient. We are also planning to compare the approach with some state-of-the art method.

Though scaling actions considered in this paper are vertical, the heuristics could be easily extended to perform horizontal scaling actions. In fact, it is sufficient to intend the involved resources as discrete, i.e., as containers or VMs with fixed capacities. In this case, the approach should be adapted so as to round up or down the identified amounts of resources to be allocated. Moreover, the proposed heuristics can be extended to manage workflow blocks that break the structure of well-formed nesting of activities, requiring to compute a makespan indicator and a (sub-optimal) resource assignment for such blocks. The heuristics could also be extended to efficiently derive the value of $\alpha \in [0, 1]$ that minimizes the makespan indicator of each block. Finally, the heuristics could also be improved by considering different performance models, so as to ensure the applicability of the method to contexts in which linearity between response time and amount of allocated resources may not be sufficient to properly characterize the behaviour of the system, e.g., due to the presence of not negligible VM start up times.

Acknowledgments. This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

References

1. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: IEEE Network Operations and Management Symposium, pp. 204–212. IEEE (2012)
2. Alshuqayran, N., Ali, N., Evans, R.: A systematic mapping study in microservice architecture. In: IEEE International Conference on SO Computing and Applications, pp. 44–51. IEEE (2016)
3. Bauer, A., Herbst, N., Spinner, S., Ali-Eldin, A., Kounev, S.: Chameleon: a hybrid, proactive auto-scaling mechanism on a level-playing field. IEEE Trans. Parallel Distrib. Syst. **30**(4), 800–813 (2018)

4. Bauer, A., Lesch, V., Versluis, L., Ilyushkin, A., Herbst, N., Kounev, S.: Chamulteon: coordinated auto-scaling of micro-services. In: IEEE International Conference on Distributed Computing Systems, pp. 2015–2025. IEEE (2019)
5. Berg, B., Dorsman, J.L., Harchol-Balter, M.: Towards optimality in parallel scheduling. Proc. ACM Meas. Anal. Comput. Syst. **1**(2), 40:1–40:30 (2017)
6. Bi, J., Zhu, Z., Tian, R., Wang, Q.: Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In: IEEE International Conference on Cloud Computing, pp. 370–377. IEEE (2010)
7. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Futur. Gener. Comput. Syst. **25**(6), 599–616 (2009)
8. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: QoS-aware replanning of composite web services. In: IEEE International Conference on Web Ser, pp. 121–129. IEEE (2005)
9. Cardellini, V., Casalicchio, E., Grassi, V., Iannucci, S., Presti, F.L., Mirandola, R.: Moses: a framework for GOS driven runtime adaptation of service-oriented systems. IEEE Trans. on Softw. Eng. **38**(5), 1138–1159 (2011)
10. Carnevali, L., Paolieri, M., Reali, R., Vicario, E.: Compositional safe approximation of response time distribution of complex workflows. In: Abate, A., Marin, A. (eds.) Quantitative Evaluation of Systems: 18th International Conference, QEST 2021, Paris, France, August 23–27, 2021, Proceedings, pp. 83–104. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85172-9_5
11. Carnevali, L., Paolieri, M., Reali, R., Vicario, E.: Compositional safe approximation of response time probability density function of complex workflows. ACM Trans. Model. Comput. Simul. (2023)
12. Carnevali, L., Reali, R., Vicario, E.: Eulero: a tool for quantitative modeling and evaluation of complex workflows. In: Ábrahám, E., Paolieri, M. (eds.) Quantitative Evaluation of Systems: 19th International Conference, QEST 2022, Warsaw, Poland, September 12–16, 2022, Proceedings, pp. 255–272. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-16336-4_13
13. Chen, T., Bahsoon, R., Yao, X.: A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems. arXiv preprint arXiv:1609.03590 (2016)
14. Chieu, T.C., Mohindra, A., Karve, A.A., Segal, A.: Dynamic scaling of web applications in a virtualized cloud computing environment. In: IEEE International Conference on e-Business Engineering, pp. 281–286. IEEE (2009)
15. Farokhi, S., Lakew, E.B., Klein, C., Brandic, I., Elmroth, E.: Coordinating CPU and memory elasticity controllers to meet service response time constraints. In: International Conference on Cloud and Autonomic Computing, pp. 69–80. IEEE (2015)
16. Fox, A., et al.: Above the Clouds: A Berkeley View of Cloud Computing. Dept. Electrical Eng. and Comput. Sci., University of California, Berkeley, Rep. UCB/EECS **28**(13), 2009 (2009)
17. Gias, A.U., Casale, G., Woodside, M.: Atom: Model-driven autoscaling for microservices. In: International Conference on Distributed Computing System, pp. 1994–2004. IEEE (2019)
18. Horváth, A., Paolieri, M., Ridi, L., Vicario, E.: Transient analysis of non-Markovian models using stochastic state classes. Perf. Eval. **69**(7–8), 315–335 (2012)

19. Incerto, E., Tribastone, M., Trubiani, C.: Combined vertical and horizontal autoscaling through model predictive control. In: Aldinucci, M., Padovani, L., Torquati, M. (eds.) Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27 - 31, 2018, Proceedings, pp. 147–159. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96983-1_11
20. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Futur. Gener. Comput. Syst. **27**(6), 871–879 (2011)
21. Lakew, E.B., Klein, C., Hernandez-Rodriguez, F., Elmroth, E.: Towards faster response time models for vertical elasticity. In: IEEE/ACM International Conference on Utility and Cloud Computing, pp. 560–565. IEEE (2014)
22. Liu, J., Zhang, S., Wang, Q., Wei, J.: Coordinating fast concurrency adapting with autoscaling for SLO oriented web applications. IEEE Trans. Parallel Distrib. Syst. **33**(12), 3349–3362 (2022)
23. Lynn, T., Rosati, P., Lejeune, A., Emeakaroha, V.: A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In: IEEE International Conference on Cloud Computing Technology and Science, pp. 162–169. IEEE (2017)
24. Nguyen, H., Shen, Z., Gu, X., Subbiah, S., Wilkes, J.: Agile: Elastic distributed resource scaling for infrastructure-as-a-service (2013)
25. Qiu, H., Banerjee, S.S., Jha, S., Kalbarczyk, Z.T., Iyer, R.K.: Firm: An intelligent fine-grained resource management framework for SLO-oriented microservices. In: USENIX Symposium on Operating Systems Design and Implementation (2020)
26. Qu, C., Calheiros, R.N., Buyya, R.: Auto-scaling web applications in clouds: a taxonomy and survey. ACM Comput. Surv. **51**(4), 1–33 (2018)
27. Rahman, J., Lama, P.: Predicting the end-to-end tail latency of containerized microservices in the cloud. In: International Conference on Cloud Engineering, pp. 200–210. IEEE (2019)
28. Rose, K., Eldridge, S., Chapin, L.: The internet of things: an overview. The internet society (ISOC) **80**, 1–50 (2015)
29. Rosenberg, F., Leitner, P., Michlmayr, A., Celikovic, P., Dustdar, S.: Towards composition as a service-a quality of service driven approach. In: IEEE International Confernce on Data Engineering, pp. 1733–1740. IEEE (2009)
30. Russell, N., Ter Hofstede, A.H., Van Der Aalst, W.M., Mulyar, N.: Workflow control-flow patterns: A revised view. BPM Center Report BPM-06-22, BPMcenter. org 2006 (2006)
31. Salah, K., Elbadawi, K., Boutaba, R.: An analytical model for estimating cloud resources of elastic services. J. of Network and Sys. Manag. **24**, 285–308 (2016)
32. Salehie, M., Tahvildari, L.: Self-adaptive software: landscape and research challenges. ACM Trans. Auton. Adapt. Syst. **4**(2), 1–42 (2009)
33. Shahrad, M., Balkind, J., Wentzlaff, D.: Architectural implications of function-as-a-service computing. In: IEEE/ACM International Symposium on microarchitecture, pp. 1063–1075 (2019)
34. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: elastic resource scaling for multi-tenant cloud systems. In: ACM Symposium on Cloud Computing, pp. 1–14 (2011)
35. Stieß, S., Becker, S., Ege, F., Höppner, S., Tichy, M.: Coordination and explanation of reconfigurations in self-adaptive high-performance systems. In: International Conference on Model Driven Engineering Languages and Systems: Companion Proc, pp. 486–490 (2022)

36. Trivedi, K.S., Sahner, R.: Sharpe at the age of twenty two. ACM SIGMETRICS Performance Eval. Rev. **36**(4), 52–57 (2009)
37. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., Wood, T.: Agile dynamic provisioning of multi-tier internet applications. ACM Trans. Auton. Adapt. Syst. (TAAS) **3**(1), 1–39 (2008)
38. Vicario, E., Sassoli, L., Carnevali, L.: Using stochastic state classes in quantitative evaluation of dense-time reactive systems. IEEE Trans. on Soft. Eng. **35**(5), 703–719 (2009)
39. Yazdanov, L., Fetzer, C.: Vertical scaling for prioritized VMs provisioning. In: International Conference on Cloud and Green Computing, pp. 118–125. IEEE (2012)
40. Zheng, Z., Trivedi, K.S., Qiu, K., Xia, R.: Semi-Markov models of composite web services for their performance, reliability and bottlenecks. IEEE Trans. Serv. Comput. **10**(3), 448–460 (2015)