






Compositional Coordinated Resource Provisioning in Workflows With Stochastic Durations

Laura Carnevali , *Member, IEEE*, Marco Paolieri , *Member, IEEE*, Riccardo Reali ,
Leonardo Scommegna , *Member, IEEE*, and Enrico Vicario , *Member, IEEE*

Abstract—In performance engineering of composed services, coordinated provisioning can reduce the amount of resources required to meet end-to-end response time objectives. To this aim, various intertwined aspects of the application architecture need to be taken into account, notably including precedence constraints in the composition of elementary services, along with their durations and sensitivity to the scaling of provisioned resources. We address coordinated provisioning of resources for elementary services with stochastic durations with general distributions (i.e., including non-exponential distributions). We compose services in a workflow where precedence constraints define a Directed Acyclic Graph (DAG) and the distribution of the end-to-end (E2E) response time is subject to a Service Level Objective (SLO). We leverage a surrogate model of service performance, assuming a low workload of workflow requests (i.e., a single-request scenario) and service durations inversely proportional to provisioned resources. Given the total amount of resources, our approach derives the service provisioning that optimizes the workflow E2E response time distribution, by exploiting a compositional approach and by using stochastically ordered precedence approximations to manage dependencies in non-well-nested precedence DAGs. Then, the approach scales provisioned resources up or down to determine the minimum amount of resources needed to satisfy the SLO, while leaving the remaining resources for horizontal scaling in order to manage multiple workflow requests at high workloads. Experiments consider low-workload and high-workload scenarios, different relations between elementary service durations and provisioned resources, and workflow topologies taken from benchmarks or randomly generated with controlled statistics, using elementary service durations from a dataset of the literature. Results show that the technique is feasible also for workflows with a thousand of services and that it outperforms other provisioning methods in fitting the SLO using the same resource amount and in minimizing the resource amount needed to fit the SLO.

Index Terms—Coordinated resource provisioning, composed services, end-to-end response time, non-Markovian durations.

Received 14 August 2024; revised 22 April 2025; accepted 16 June 2025. Date of publication 3 July 2025; date of current version 31 July 2025. This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, and in part by the “Telecommunications of the Future” under Grant PE00000001 - program “RESTART”. Recommended for acceptance by B. Ucar. (*Corresponding author: Enrico Vicario.*)

Laura Carnevali, Riccardo Reali, Leonardo Scommegna, and Enrico Vicario are with the Department of Information Engineering, University of Florence, 50139 Firenze, Italy (e-mail: laura.carnevali@unifi.it; riccardo.reali@unifi.it; leonardo.scommegna@unifi.it; enrico.vicario@unifi.it).

Marco Paolieri is with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: paolieri@usc.edu).

Digital Object Identifier 10.1109/TPDS.2025.3585821

ACRONYMS AND SYMBOLS

BDDP	Balanced Duration Driven Provisioning.
CAA	Completely Agnostic Accounting.
CAP	Completely Agnostic Provisioning.
CDF	Cumulative Distribution Function.
DAG	Directed Acyclic Graph.
EUA	Effective Utilization Accounting.
E2E	end-to-end
IaaS	Infrastructure as a Service.
PDF	Probability Density Function.
QoS	Quality of Service.
RLT	Repeated Leaves Tree.
RPA	Resource Proportional Accounting.
SDP	Sensitivity Driven Provisioning.
SLO	Service Level Objective.
TDP	Topology Driven Provisioning.
UDDP	Unbalanced Duration Driven Provisioning
$E[X]$	expected value of the random variable X .
J	job size of a service (see (3)).
P	function defining provisioning of RLT nodes.
\mathcal{P}	function defining provisioning of the sets of replicated leaves of an RLT.
R_{SLO}	estimate on the minimum resource amount needed to satisfy the workflow SLO X_{SLO} (see (5)).
$T(R)$	response time of a service provisioned with a resource amount R (see (3)).
X_{SLO}	SLO on the E2E response time of a workflow (see Definition 3).
Z	job size proxy of a service (see (4)).
α	hyper-parameter defining the resource provisioning in fork-join constructs (see Section V-D3).
$\beta(X, Y)$	lower bound on $\iota(X, Y)$ (see (13)).
$\delta(X, Y)$	pairwise comparison dominance between two random variables X and Y (see Definition 1).
ϵ	threshold on the compliance of the workflow response time with its SLO (see Definition 3).
$\iota(X, Y)$	relative improvement of a random variable X over a random variable Y (see (12)).

I. INTRODUCTION

In cloud computing systems, effective resource provisioning plays a crucial role for efficient achievement of Service Level Objectives (SLOs). While cloud providers and orchestrators perform actuation, service developers are

responsible for specifying the contract driving the run-time allocation. This task is particularly hard for composed services, where efficient usage of resources requires balance among individual services.

The problem subtends two orthogonal dimensions, with horizontal scaling supporting the required concurrency degree in service delivery, and vertical scaling managing demand of non-compressible resources and execution time of each service instance. We focus on the latter aspect, addressing composed services subject to requirements on the end-to-end (E2E) response time, which assumes relevance in various technological spaces, including computationally intensive containerized services and cloud native Functions as a Service (FaaS) [1] with limited provisioning budget, Network Functions (NF) and applications subject to requirements on low latency, edge-to-cloud applications with limited computing capabilities at edge or near-edge Multi-access Edge Computing (MEC) nodes [2], [3], and scientific workflows requiring high computation power [4], [5]. In these scenarios, coordination provides a significant margin of optimization in resource provisioning but requires support to explore the design space, capturing together duration and job size of elementary services with precedence constraints arising in their composition within a workflow.

In this paper, we present a solution method for coordinated provisioning of resources for elementary services, considering CPU resources of the same type on the same machine. We compose services in a complex workflow with the topology of a Directed Acyclic Graph (DAG). Services require resources allocated exclusively, as in an Infrastructure as a Service (IaaS) environment subject to structural, technological, or real-time requirements (e.g., avoiding long high restart times in composed services subject to strict low latency requirements); in particular, we do not address resource contention issues that occur in Platform as a Service (PaaS) [6] or FaaS [7] environments. Services have durations characterized by general distributions (i.e., including non-exponential distributions with possibly bounded supports), facilitating fitting of measured response times and representation of real-time constraints. The workflow is subject to a Service Level Objective (SLO) on its E2E response time distribution, supporting the definition of *soft deadlines* and *penalty functions* [8], [9], which can be expressed as rewards calculated from the E2E response time distribution.

We exploit a surrogate performance model with low complexity, assuming that the workflow is subject to a low workload (i.e., each workflow request is served immediately as in a single-request scenario) and that durations of services scale with provisioned resources according to a homogeneous linear relation [10], [11], [12]. In this setting, given the total amount of resources, we derive the service provisioning that optimizes the workflow E2E response time distribution, by exploiting a compositional method and by adopting stochastically ordered approximations to manage dependencies in non-well-nested topologies. We estimate the minimum amount of resources needed to satisfy the workflow SLO in terms of *pairwise comparison dominance*, leaving the remaining resources available to manage multiple workflow requests at high workloads through horizontal scaling (which assigns to each additional service replica the resource amount defined by our method).

We perform experiments for workflow topologies taken from benchmarks or randomly generated with controlled statistics, using elementary service durations from a dataset of the literature. Results demonstrate that our technique is feasible also for workflows with thousands of services and that it outperforms alternative provisioning methods in fitting the workflow SLO, both in low-workload and in high-workload scenarios, even when scaling of service durations is not homogeneous linear.

To summarize, our main contributions are:

- An approach for coordinated allocation of resources to services of complex workflows with DAG topology.
- Characterization of service durations as general distributions over bounded support, and of the workflow SLO as soft deadline on the E2E response time distribution.
- Application of the pairwise comparison dominance as a measure of the achievement of the workflow SLO.
- A thorough experimentation assessing feasibility and effectiveness of the proposed approach.
- An open-source artifact¹ that supports replication of the experimental results (released under the AGPLv3 license).

The rest of the paper is organized as follows. Section II discusses related works and Section III provides an overview of the proposed approach. Then, Section IV specifies the considered class of workflows, Section V presents the proposed solution method, Section VI illustrates the experimental methodology, and Section VII presents the obtained results. Finally, Section VIII draws conclusions, providing a discussion on the limitations of the approach and on future work.

II. RELATED WORKS

The literature on service-oriented systems [13], cloud computing systems [14], and edge/fog computing systems [15] has widely addressed the issue of resource scaling. In the framework of [13], which characterizes the problem of adaptation of service-oriented architectures to fulfill non-functional Quality of Service (QoS) requirements, we are interested in investigating state-of-the-art approaches from three main perspectives: *a) the composition structure of services*, i.e., whether adaptation addresses single services or multiple services composed in different topologies; *b) the goal of resource adaptation*, i.e., the QoS metrics (e.g., performance, reliability, cost) to be optimized; *c) the actions of resource adaptation*, with specific focus on *coordination* mechanisms, i.e., which coordinated operations are performed in the adaptation of resources.

a) Composition Structure of Services: Scaling methods have addressed different typologies of workflows of services. In [10], [16], [17], queuing networks model a single cloud service with exponential arriving and service times, enabling prediction of SLO violations and triggering of scaling actions. The majority of works consider *n*-tier applications [11], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], which are merely modeled as sequential queuing networks, enabling characterization of the performance and incoming traffic of the application tiers, and identification of performance bottlenecks. Application of machine learning or estimation theory

¹<https://doi.org/10.5281/zenodo.15754793>

approaches estimate queuing network parameters from previous observations of the system, e.g., time series analysis [18] and expectation-maximization [22]. To counteract performance inefficiencies, queuing networks can be solved or integrated with control theory approaches [20], [31], rule-based agents [29], machine learning techniques [18], [24], [32], or combinations of these methods [26]. To the best of our knowledge, only a few approaches address provisioning in workflows with more complex topologies, as emerging in microservices architectures [6], [33], [34], though considering compositions of at most a few dozen services due to the high complexity of managing multiple inter-service dependencies. The literature on workflow scheduling [35], [36] has widely addressed service compositions with DAG topology, though solving the different problem of assigning shared virtual resources for specific time intervals to the tasks of a single workflow instance, rather than adapting the provisioning of private (i.e., exclusively allocated) virtual resources to the tasks based on the QoS requirements of a flow of workflow requests. Workflow scheduling has been investigated both in multiprocessor systems [37] and in distributed infrastructures such as grids [38], using a variety of scheduling approaches such as evolutionary algorithms [38], [39] as well as heuristic [40], [41] and metaheuristic methods [42], and considering different non-functional requirements in the scheduling objective, such as minimizing both makespan and resource costs [43], [44].

b) Goal of Adaptation: Autonomic scaling uses different metrics to detect SLO violations. Many approaches consider the E2E response time as the QoS indicator of an application [10], [11], [16], [18], [20], [21], [22], [24], e.g., the average response time [10], [20], [24], [45], [46], the maximum response time [16], or percentiles of the response time [21], [22], [26]. Other approaches implement strategies based on hybrid performance indicators, a valuable option to prevent cases of over-provisioning that typically arise when partial information on the workflow is used for resource scaling. In such cases, the E2E response time is evaluated, alternatively or in addition to, resource utilization [6], [17], [26], [27], [29], [31], [34], [45], costs [23], [47], energy power [48], or throughput of served requests [30], [33]. As a common trait, all these approaches define SLOs as thresholds on expected metrics rather than as distributions of the random variables representing the metrics.

c) Coordinated Adaptation Actions: Coordination mechanisms that consider multiple aspects of complex web applications can increase resource utilization while satisfying SLOs on the E2E response time. In [27], [30], both over- and under-provisioning are solved by coordinating scaling of hardware and software resources (e.g., threads, database connections), which typically causes inefficiencies in the concurrent processing of requests in layers of n -tier applications. Vertical and horizontal scaling are coordinated in [33], [34]. Model Predictive Control (MPC) is used in [33] to iteratively optimize performance and resource utilization objectives, solving a constrained non-linear optimization problem defined on the fluid approximation of a Layered Queuing Network (LQN). In [34], a method estimates performance of a microservices architecture by exploring the LQN space to find the scaling strategy that maximizes the system revenue while reducing the amount of provisioned resources. In [6], critical paths of a microservices architecture

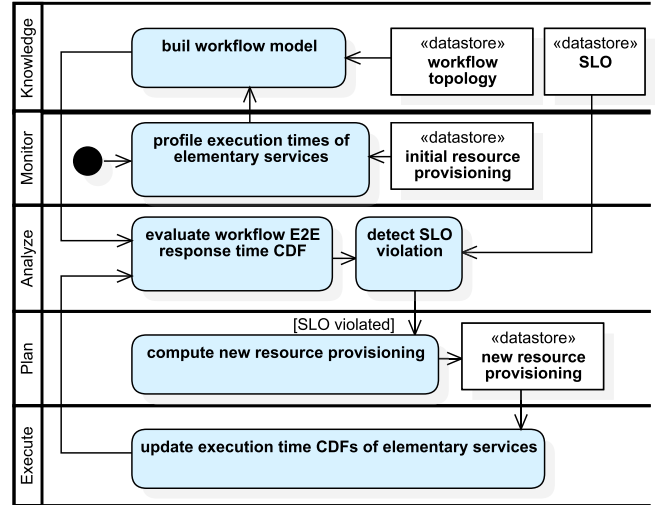


Fig. 1. A MAPE-K loop diagram of the proposed approach.

are detected by an approach based on support vector machines, identifying services that violate SLOs and re-provisioning them to meet the E2E response time SLO, maximizing resource utilization and mitigating resource contention. In [49], an approach performs coordinated resource provisioning of workflows with well-nested topology to meet an SLO on the E2E response time distribution, though performing preliminary experiments only on a small synthetic workflow.

d) Contribution: In the framework of [13], our approach is defined by the following attributes: the *composition structure of services* features complex DAG topologies, with hundreds up to thousands of services having non-Markovian durations possibly over bounded supports; the *goal of resource adaptation* is to ensure the fulfillment of non-functional requirements on the workflow E2E response time distribution; the *adaptation actions* consist of service tuning operations that change the amount of resources provisioned to individual services in a coordinated manner, which, in turn, changes the Cumulative Distribution Function (CDF), or equivalently the Probability Density Function (PDF), of the response time of individual services, thus changing also the workflow response time CDF.

III. SOLUTION OVERVIEW

Fig. 1 shows the MAPE-K loop (Monitor-Analyze-Plan-Execute over a shared Knowledge) [50] of our approach. In the *Knowledge* phase, we model a workflow of activities [51] by a UML activity diagram (see Section IV). To this end, we derive the response time CDF of each elementary service by repeatedly measuring its duration for *any* given resource assignment during the *Monitor* phase, and by fitting samples using GEN exponential distributions [52], [53].

In the *Analyze* phase, we efficiently evaluate the CDF of the E2E response time of the entire workflow using the approach of [54]. Then, we detect violation of the SLO according to the *pairwise-comparison dominance* (see Section V-A).

In the *Plan* phase, we define a surrogate performance model with low complexity, where durations of elementary services scale linearly with provisioned resources, and queue effects are

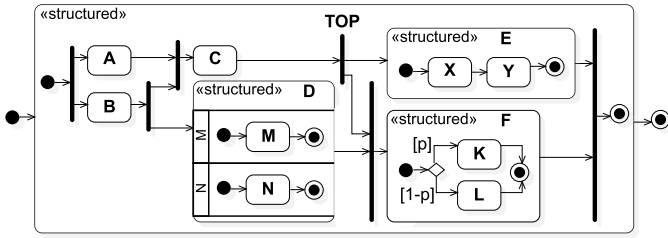


Fig. 2. UML activity diagram of a workflow.

not considered as in low-workload scenarios (see Section V-B). We use this performance model to efficiently evaluate a resource assignment for each service (Sections V-C, V-D, V-E, and V-F), addressing both well-nested and non-well-nested compositions of services.

In the *Execute* phase, we update the CDF of each elementary service based on provisioned resources, and we evaluate the workflow E2E response time CDF restarting the loop in Fig. 1.

Note that the workflow topology and the response time CDF of each elementary service under *any* given resource allocation comprise the only prerequisite information to run the approach. In fact, profiling the response time of each elementary service for a single resource allocation is sufficient to estimate the service job size in our surrogate performance model. Thus, profiling of service response times under different resource provisioning is not needed to execute our approach.

IV. WORKFLOWS WITH STOCHASTIC SERVICE DURATIONS

We represent workflows using UML activity diagrams, a widely adopted graphical representation [55], [56]. As shown in Fig. 2, a workflow consists of *elementary actions*, modeling elementary activities (e.g., A, B, and C), and *structured actions*, modeling a set of (elementary or structured) actions composed either: *i*) in a *sequence* (e.g., E models the sequential execution of X and Y), or *ii*) by a *fork-join* operator (e.g., D models the concurrent execution of M and N), or *iii*) by a *split-merge* operator (e.g., F models the exclusive execution of either K or L with probability p or $1-p$, respectively), or *iv*) in the topology of an *irreducible DAG*, i.e., a DAG that includes (separate) fork and join operators and that cannot be represented with sequence and fork-join operators² (e.g., TOP composes A, B, C, D, E, and F in the topology of an irreducible DAG).

Note that sequence, fork-join, and split-merge operators identify a specific topology. Conversely, irreducible DAG constructs identify a countably infinite set of topologies, each regarded here as a type of composition. A workflow that does not include irreducible DAG constructs is termed *well-nested*. Otherwise, a workflow is termed *non-well-nested*.

We define the semantics of a workflow in terms of the random variable $\tau(a)$ modeling the E2E response time of each (elementary or structured) action a of the activity diagram:

- If a is an elementary action provisioned with resources R_a , $\tau(a)$ is the random variable $T_a(R_a)$ modeling the duration of a when its execution is provisioned with resources R_a .

²Any DAG topology with less than four (elementary or structured) actions is always reducible to sequence and fork-join operators.

- If a is a sequence of (elementary or structured) actions a_1, \dots, a_n , then $\tau(a) = \sum_{i=1}^n \tau(a_i)$ is the sum of the E2E response times of the actions constituting a .
- If a is a fork-join of (elementary or structured) actions a_1, \dots, a_n , then $\tau(a) = \max_{i=1}^n \tau(a_i)$ is the maximum of the E2E response times of the actions constituting a .
- If a is a split-merge of (elementary or structured) actions a_1, \dots, a_n having probability p_1, \dots, p_n , respectively with $\sum_{i=1}^n p_i = 1$, then $\tau(a) = \tau(a_i)$ with probability $p_i \forall i \in \{1, \dots, n\}$, meaning that $\tau(a)$ is a mixture of the E2E response time PDFs of the actions constituting a .
- If a is a composition of (elementary or structured) actions a_1, \dots, a_n with irreducible DAG topology, then $\tau(a)$ is recursively evaluated by a DAG backward traversal. We define the *exit time* $T_e(a_i)$ of action a_i as the E2E response time of a_i itself plus the maximum among the exit times of its preceding actions v_1, \dots, v_n , i.e., $T_e(a_i) = \tau(a_i) + \max_j T_e(v_j)$ (e.g., $T_e(C) = \tau(C) + \max\{T_e(A), T_e(B)\}$), where the exit time of an action with no preceding action is its E2E response time (e.g., $T_e(A) = \tau(A)$). Given that a DAG topology terminates either with an action or with a join of actions, the E2E response time of a is either the exit time of its last action (if any) or the maximum among the exit times of the input actions z_1, \dots, z_m of its last join (e.g., $T(\text{TOP}) = \max\{T_e(E), T_e(F)\}$).

V. SOLUTION METHOD

In this section, first we formulate the problem of compliance with an SLO on the workflow E2E response time CDF (Section V-A). Then, we define our surrogate model of service performance and we estimate the minimum amount of resources needed to meet the workflow SLO (Section V-B). Finally, we illustrate a hierarchical formalism for workflow representation (Section V-C) and we present our resource provisioning technique and its accounting heuristics (Sections V-D, V-E, and V-F).

A. SLO Compliance Problem

We determine the compliance of a workflow with its SLO by the pairwise-comparison dominance between the SLO and the E2E response time distribution of the workflow.

Definition 1 (Pairwise-Comparison Dominance): Given two random variables X and Y that have PDFs $f_X(t)$ and $f_Y(t)$, respectively, and CDFs $F_X(t)$ and $F_Y(t)$, respectively, we define the pairwise comparison dominance $\delta(X, Y)$ between X and Y as the probability that X is not larger than Y :

$$\delta(X, Y) := \text{Prob}\{X \leq Y\} = \int_0^{\infty} (1 - F_Y(t)) \cdot f_X(t) dt. \quad (1)$$

If $\delta(X, Y) \geq \frac{1}{2}$, X is said to be lower than Y according to pairwise-comparison dominance, which is denoted as $X \preceq Y$.

Note that the pairwise-comparison dominance between two random variables X and Y is a relaxation of *stochastic ordering* between X and Y , which occurs if $F_X(t) \geq F_Y(t) \forall t$, while $\delta(X, Y)$ provides an average measure of how many times X anticipates Y . Also note that, if X and Y have bounded supports, $\delta(X, Y)$ may subtend a relation of *deterministic ordering*,

which occurs if the supports of X and Y are disjoint, i.e., if either $\text{Prob}\{X \leq Y\} = 1$ or $\text{Prob}\{Y \leq X\} = 1$. However, in this case, $\delta(X, Y)$ does not quantify the distance between the supports of X and Y , i.e., it does not provide a measure of how much X anticipates Y or viceversa.

Definition 2 (Pairwise-Comparison Equivalence): Two random variables X and Y are pairwise-comparison equivalent, which is denoted as $X \equiv Y$, if $\delta(X, Y) = \delta(Y, X)$, i.e., if $\int_0^\infty (1 - F_Y(t)) \cdot f_X(t) dt = \int_0^\infty (1 - F_X(t)) \cdot f_Y(t) dt = \frac{1}{2}$.

We define a workflow to be compliant with its SLO if its E2E response time is equivalent to the SLO up to a tolerance:

Definition 3 (SLO Compliance): A workflow having E2E response time X is termed compliant with an SLO X_{SLO} if $|\delta(X, X_{\text{SLO}}) - \frac{1}{2}| \leq \epsilon$, where X_{SLO} is the random variable modeling the SLO time, $\epsilon > 0$ is an arbitrary threshold, and $|\delta(X, X_{\text{SLO}}) - \frac{1}{2}|$ is termed *dominance deviation*.

Therefore, according to Definitions 1, 2, and 3, given a workflow composing elementary services a_1, \dots, a_n with provisioned resources R_1, \dots, R_n , respectively, our optimization problem consists in minimizing the total amount of resources while guaranteeing SLO compliance within some threshold $\epsilon > 0$:

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^n R_i \\ & \text{subject to} && \left| \delta(X, X_{\text{SLO}}) - \frac{1}{2} \right| \leq \epsilon \end{aligned} \quad (2)$$

Note that other stochastic orders between X and X_{SLO} could be considered in Definition 3 and thus in (2), without affecting the complexity of our resource provisioning technique.

B. Surrogate Model of Service Performance

We define a surrogate model of service performance, achieving low complexity by means of two simplifying assumptions:

- *Hypothesis 1:* The workflow is subject to a low workload.
- *Hypothesis 2:* Service durations scale with provisioned resources according to a homogeneous linear relation.

Because of Hypothesis 1, we assume that each service request is served immediately as soon as it is issued, without queuing effects. Thus, we actually consider a single-request scenario.

We model the *job size* J of an elementary or structured action of the workflow as the time required to complete it with a unitary amount of resources [57], [58]. Because of Hypothesis 2, the response time $T(R)$ of the action when using a given amount R of provisioned resources (i.e., the total resources provisioned to elementary services composed in the action) is equal to

$$T(R) := \frac{J}{R}. \quad (3)$$

Since J is a random variable, $T(R)$ is also a random variable. Notably, we assume that the job size is invariant of the amount of provisioned resources; therefore, profiling response times for a single resource provisioning is sufficient to estimate the PDF of the job size in our surrogate performance model (as remarked in Section III) and thus to derive the response time PDF for different resource allocations using (3).

In the experiments of Sections VI and VII, we model the job size of each elementary service by a shifted and truncated exponential random variable, fitting the mean value and the coefficient of variation [59] of measured service durations $T(R)$ (obtained from [60]) multiplied by the amount of allocated resources R , i.e., J is a random variable having support $[a, b]$, PDF $f(t) := \lambda e^{-\lambda t} e^{a\lambda} / (1 - e^{(a-b)\lambda})$ for some rate $\lambda > 0$, and CDF $F(t) := (1 - e^{a\lambda} e^{-\lambda t}) / (1 - e^{(a-b)\lambda})$. According to (3), $T(R)$ is also a shifted and truncated exponential random variable with scaled support $[a/R, b/R]$ and rate λR .

Note that, for services with durations depending on the request payload (e.g., database access with duration depending on the database size, data sorting with duration depending on the number and values of data), the job size distribution could model data dependency if the distribution of data were known; profiling of the job size with input data extracted from such distribution could be performed to estimate this data-dependent job size distribution and, in turn, to model the response time for any given resource provisioning, as already remarked.

In the exploration of resource allocations, to reduce complexity due to the fact that $T(R)$ and J are random variables, we consider the expected value $E[T(R)]$ as a scalar proxy, defined as the ratio of a scalar $E[J]$ (the expected job size, i.e., our *job size proxy*) and the resource amount R :

$$E[T(R)] = \frac{E[J]}{R}. \quad (4)$$

According to (3), the job size proxy $E[J]$ is also invariant with respect to the provisioned resources: it can be estimated as $R \cdot E[T(R)]$ from the mean response time $E[T(R)]$ measured for a given resource assignment R ; then, (4) can estimate $E[T(R')] = E[J]/R'$ for an arbitrary amount of resources R' . While we use the job size proxy to keep the complexity of our surrogate performance model low, our approach derives the expected value of the response time of each (elementary or composed) service from the full E2E response time distribution.

Finally, given a workflow (i.e., a top activity) with provisioned resources R and expected response time $E[T(R)]$, we use (4) to estimate the minimum resources R_{SLO} that satisfy an SLO X_{SLO} with expected value $E[X_{\text{SLO}}]$:

$$R_{\text{SLO}} = \frac{E[J]}{E[X_{\text{SLO}}]} = \frac{R \cdot E[T(R)]}{E[X_{\text{SLO}}]}. \quad (5)$$

C. Hierarchical Workflow Model

To allocate resources, we model workflows using Repeated Leaves Trees (RLTs), i.e., trees that include replicas of elementary actions of irreducible DAGs for each different path in which they appear.

Definition 4 (Repeated Leaves Tree): An RLT is a tuple $\text{RLT} := \langle V, E, v_0, H \rangle$ whose elements are defined as follows.

- $V = N \cup L$, with $N \cap L = \emptyset$, is the set of vertices, partitioned into the sets of internal nodes N (e.g., E in Fig. 3(b)) and leaf nodes L (e.g., X). An internal node models a structured action and is labeled with SEQ, AND, or XOR depending on whether the structured action composes other

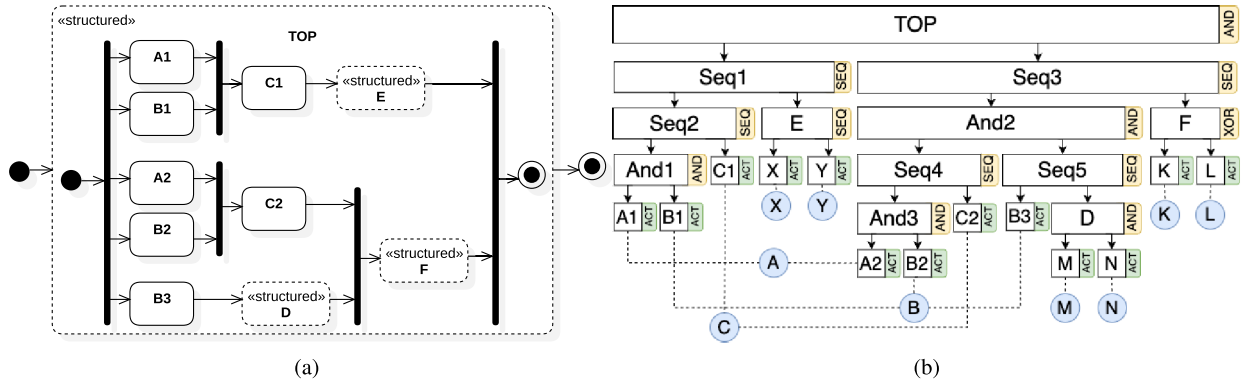


Fig. 3. (a) UML activity diagram of a variant of the workflow of Fig. 2, obtained by replicating the elementary actions belonging to different paths of the top-level irreducible DAG (the elementary actions composing the structured actions D, E, and F are not shown). (b) RLT of the workflow of Fig. 2, obtained from the activity diagram of Fig. 3(a) (sets of replicated elementary actions are depicted as blue circled nodes, with dashed lines connecting the action replicas).

actions with the sequence, fork-join, or split-merge construct, respectively. A leaf node models and elementary action and is labeled with ACT.

- $E \subseteq V \times V$ is the set of edges, encoding the parent-child relation of workflow activities (e.g., arrows from node E to nodes X and Y show that E contains these activities).
- $H = \{h_1, \dots, h_n\}$ is a partition of the leaves L , i.e., $\cup_{i=1}^n h_i = L$ and $h_i \cap h_j = \emptyset$ for $i \neq j$, such that h_i contains all the replicas of the elementary activity a_i of the workflow, for $i = 1, \dots, n$. Graphically, we represent the sets h_i as circled nodes connecting, with dashed lines, the leaves that they contain (e.g., in Fig. 3(b), the set B contains leaves B1, B2 and B3).
- $v_0 \in V$ is the root node (e.g., TOP), the unique node with no parents, i.e., $\exists! r \in V \mid \#(n, r) \in D \forall n \in N$.

We derive the RLT from the UML activity diagram of a variant of the workflow obtained by replicating each elementary action that belongs to multiple paths of an irreducible DAG construct, yielding a different instance of the action for each path. To this end, we perform a DAG backward traversal where: we replicate each (elementary or structured) action that is the predecessor of more than one action; we replicate predecessors of a replicated action; we replace replicated structured actions by the composition of their (replicated) elementary actions. For example, we translate the workflow of Fig. 2 into the one of Fig. 3(a) by performing a backward traversal of the top-level DAG: we do not replicate E and F, which are not predecessors of an action; we do not replicate D, which is the predecessor of a single action (i.e., F); we replicate C into C1 and C2 given that it is the predecessor of E; we replicate A into A1 and A2 given that it is the predecessor of the replicated action C; and, we replicate B into B1, B2, and B3 given that it is the predecessor of the predecessor of D and of the replicated action C.

Then, we easily derive the RLT of a workflow from the obtained activity diagram by mapping elementary actions into leaf nodes, and sequential, fork-join, and split-merge constructs into internal nodes with the corresponding label SEQ, AND, and XOR, respectively. Fig. 3(b) shows the RLT of the workflow of Fig. 2, obtained from the activity diagram of Fig. 3(a). Also note that the RLT is a special case of a directed hypergraph [61] where hyperedges are used to identify leaves that are replicas of the same action.

D. Resource Provisioning for the Hierarchical Workflow Model

We consider a resource provisioning $P : V \rightarrow \mathbb{R}_{>0}$ assigning resources to each node $v \in V$ of the RLT $\langle V, E, v_0, H \rangle$ so that the amount of each internal node $v \in N$ is the sum of the amounts of its child nodes, i.e., $P(v) = \sum_{v_i \mid (v, v_i) \in E} P(v_i)$. Given a resource amount R provisioned for the workflow, we define a new provisioning $P' : V \rightarrow \mathbb{R}_{>0}$ partitioning R among the leaf nodes of the RLT. To this end, we perform Algorithm 1, which executes a *top-down* visit of the RLT. Specifically:

- We assign the resource amount R to the root node v_0 , modeling the whole workflow (line 1), i.e., $P'(v_0) = R$.
- At each step, we consider an internal node $v \in N \subset V$ provisioned with a resource amount $P'(v)$ (lines 3-4); at the initial step, node v is the root node v_0 (line 2). We partition the amount $P'(v)$ among the child nodes of node v (lines 8-14) with the aim of minimizing the expected response time $E[T(P'(v))]$ of v . To this end:
 - We express $E[T(P'(v))]$ as a function of the expected response times $E[T_i(P'(v_i))]$ of its child nodes v_i . In turn, $E[T_i(P'(v_i))] = E[J_i]/P'(v_i) = E[T_i(P(v_i))]P(v_i)/P'(v_i)$ according to (4) (line 7). We derive $E[T_i(P(v_i))]$ from the response time CDF of node v_i with provisioning $P(v_i)$ (line 6), which, in turn, can be evaluated from the CDFs of the response time random variables $\tau(v), v \in L$ by using the approach of [62] (line 5).
 - We define the amount $P'(v_i)$ of each child node v_i of v differently depending on whether v composes its child nodes through sequence (lines 9-10), split-merge (lines 11-12), or fork-join (lines 13-14) constructs.

1) *Sequence (SEQ)*: Let node v be the sequence of nodes v_1, v_2, \dots, v_N . The resource amounts $P'(v_i)$ that minimize the expected response time $E[T(P'(v))]$ of node v are:

$$P'(v_i) = \frac{\sqrt{E[J_i]}}{\sum_{j=1}^N \sqrt{E[J_j]}} \cdot P'(v) \quad \forall i \in \{1, 2, \dots, N\}. \quad (6)$$

Let $R_i := P'(v_i)$ and $Z_i := E[J_i] \forall i \in \{1, 2, \dots, N\}$. By (4), $E[T(P'(v))] = \sum_{i=1}^N E[T_i(R_i)] = \sum_{i=1}^N Z_i/R_i$. To minimize $\sum_{i=1}^N Z_i/R_i$ subject to $\sum_{i=1}^N R_i = P'(v)$, we

Algorithm 1: Evaluate Resource Provisioning for the RLT.

VisitRLT($\langle V, E, v_0, H \rangle, P, R, \tau$)
input : RLT $\langle V, E, v_0, H \rangle$, provisioning $P : V \rightarrow \mathbb{R}_{>0}$,
workflow resource amount R , response time
random variables $\tau(v), v \in L$ for provisioning P
output : new provisioning $P' : V \rightarrow \mathbb{R}_{>0}$

- 1 $P'(v_0) \leftarrow R$
- 2 $Q \leftarrow \{v_0\}$
- 3 **while** Q is not empty **do**
- 4 Select and remove a node v from Q
- 5 Derive the CDF of the response time $T_i(P(v_i))$ of v_i
with provisioning $P(v_i)$ from τ by using [62]
- 6 Derive the expected response time $E[T_i(P(v_i))]$ of v_i
- 7 Derive the job size proxy $E[J_i]$ of v_i using Eq. (4)
- 8 **foreach** child node v_i of v **do**
- 9 **if** v is a SEQ node **then**
- 10 | compute $P'(v_i)$ according to Eq. (6)
- 11 **if** v is an XOR node **then**
- 12 | compute $P'(v_i)$ according to Eq. (7)
- 13 **if** v is an AND node **then**
- 14 | compute $P'(v_i)$ according to Eq. (8)
- 15 **if** v_i is an internal node (i.e., $v_i \in N$) **then**
- 16 | $Q \leftarrow Q \cup \{v_i\}$
- 17 **return** P'

use the Lagrangian $\mathcal{L}(R_1, \dots, R_N, \lambda) = \sum_{i=1}^N Z_i/R_i + \lambda(\sum_{i=1}^N R_i - P'(v))$, and obtain $\partial\mathcal{L}/\partial R_i = -Z_i/R_i^2 + \lambda = 0$ for $i = 1, \dots, N$ (hence, $R_i = \sqrt{Z_i}/\sqrt{\lambda}$) and $\partial\mathcal{L}/\partial\lambda = \sum_{i=1}^N R_i - P'(v) = 0$ (hence, $\sum_{i=1}^N \sqrt{Z_i} = \sqrt{\lambda}P'(v)$). These equations imply that $R_i = (\sqrt{Z_i}/\sum_{j=1}^N \sqrt{Z_j})P'(v)$, which corresponds to the objective $(\sum_{i=1}^N \sqrt{Z_i})^2/P'(v)$.

Eq. (6) indicates that allocating resources equally among the child nodes of v (i.e., $P'(v_i) = P'(v)/N \forall i \in \{1, 2, \dots, N\}$) does not minimize the expected response time of v . Instead, the sequence exhibits a different *sensitivity* to resource variations of its composed activities, which is proportional to the square root of their expected job sizes.

2) *Split-Merge* (XOR): Let node v be the split-merge of nodes v_1, v_2, \dots, v_N , with probabilities p_1, p_2, \dots, p_N , respectively. The resource amounts $P'(v_u)$ that minimize the expected response time $E[T(P'(v))]$ of node v are:

$$P'(v_i) = \frac{\sqrt{p_i E[J_i]}}{\sum_{j=1}^N \sqrt{p_j E[J_j]}} \cdot P'(v) \quad \forall i \in \{1, 2, \dots, N\}. \quad (7)$$

Let $R_i := P'(v_i)$ and $Z_i := E[J_i] \forall i \in \{1, 2, \dots, N\}$. From (4), $E[T(P'(v))] = \sum_{i=1}^N p_i E[T_i(R_i)] = \sum_{i=1}^N p_i Z_i/R_i$. To minimize $\sum_{i=1}^N p_i Z_i/R_i$ subject to $\sum_{i=1}^N R_i = P'(v)$, we use the Lagrangian $\mathcal{L}(R_1, \dots, R_N, \lambda) = \sum_{i=1}^N p_i Z_i/R_i + \lambda(\sum_{i=1}^N R_i - P'(v))$ and obtain $\partial\mathcal{L}/\partial R_i = -p_i Z_i/R_i^2 + \lambda = 0$ for $i = 1, \dots, N$ (hence, $R_i = \sqrt{p_i Z_i}/\sqrt{\lambda}$) and $\partial\mathcal{L}/\partial\lambda = \sum_{i=1}^N R_i - P'(v) = 0$ (hence, $\sum_{i=1}^N \sqrt{p_i Z_i} = \sqrt{\lambda}P'(v)$). These equations imply $R_i = (\sqrt{p_i Z_i}/\sum_{j=1}^N \sqrt{p_j Z_j})P'(v)$, which corresponds to the objective $(\sum_{i=1}^N \sqrt{p_i Z_i})^2/P'(v)$.

We split the resource amount $P'(v)$ among v_1, v_2, \dots, v_N , managing each fraction as reserved for a service and accrued independently of whether the service actually executes. The case where resources are consumed only if the associated service

executes (e.g., in FaaS instead of IaaS) can be modeled by determining the total amount of resources as $P'(v) = \sum_{i=1}^N p_i P'(v_i)$, thus leading to a different optimal allocation.

3) *Fork-Join* (AND): Let node v be the fork-join of nodes v_1, v_2, \dots, v_N . We cannot derive the resource amounts $P'(v_i)$ that minimize the expected response time $E[T(P'(v))]$ of v for $i = 1, 2, \dots, N$, due to the fact that $E[T(P'(v))]$ cannot be expressed as a function of the expected response times $E[T_i(P'(v_i))]$ of v_1, v_2, \dots, v_N , and thus neither as a function of $P'(v_i)$ for $i = 1, 2, \dots, N$. Hence, we provide a heuristic evaluation of $P'(v_i)$, depending on a hyperparameter $\alpha \in [0, 1]$ modulating the impact of the assigned resources. Specifically, we minimize $\max(E[J_1]/(P'(v_1))^{\alpha+1}, \dots, E[J_N]/(P'(v_N))^{\alpha+1})$ subject to $\sum_{i=1}^N P'(v_i) = P'(v)$, which is achieved for

$$P'(v_i) = \frac{\alpha+1 \sqrt{E[J_j]}}{\sum_{j=1}^N \alpha+1 \sqrt{E[J_j]}} \cdot P'(v) \quad \forall i \in \{1, 2, \dots, N\}. \quad (8)$$

Let $R_i := P'(v_i)$ and $Z_i := E[J_i] \forall i \in \{1, 2, \dots, N\}$. We represent the objective using the auxiliary variable $M \geq Z_i/R_i^{\alpha+1}$ for $i = 1, \dots, N$, which is equivalent to $R_i \geq (Z_i/M)^{\frac{1}{\alpha+1}}$. Since $\sum_{i=1}^N R_i = P'(v)$, we have $P'(v) \geq \sum_{i=1}^N (Z_i/M)^{\frac{1}{\alpha+1}}$ and $M \geq (\sum_{i=1}^N Z_i^{\frac{1}{\alpha+1}}/R)^{\alpha+1}$, which gives the minimum value for the objective M . Substituting that in $R_i \geq (Z_i/M)^{\frac{1}{\alpha+1}}$, we obtain $R_i \geq P'(v) \cdot Z_i^{\frac{1}{\alpha+1}}/\sum_{i=1}^N Z_i^{\frac{1}{\alpha+1}}$, where the equality holds because of $\sum_{i=1}^N R_i = P'(v)$.

(8) balances the arguments of $\max(E[J_1]/(P'(v_1))^{\alpha+1}, \dots, E[J_N]/(P'(v_N))^{\alpha+1})$, i.e., $\forall i, j \in \{1, 2, \dots, N\}$, we have $E[J_i]/(P'(v_i))^{\alpha+1} = E[J_j]/(P'(v_j))^{\alpha+1}$. Different values of α subtend different interpretations of the heuristics:

- If $\alpha = 0$, the heuristics balances the expected response times of the child nodes, e.g., for 2 child nodes, we imposes $E[J_1]/R_1 = E[J_2]/R_2$, which, by (4), is equivalent to imposing $E[T_1(R_1)] = E[T_2(R_2)]$.
- If $\alpha = 1$, the heuristics balances the first derivatives of the expected response times of the child nodes with respect to the provisioned resources, e.g., for 2 child nodes, letting $R_i := P'(v_i)$, we impose $E[J_1]/R_1^2 = E[J_2]/R_2^2$, which, by (4), is equivalent to imposing

$$\frac{d}{dR_1} E[T_1(R_1)] = \frac{d}{dR_2} E[T_2(R_2)].$$

Therefore, the solution balances the sensitivity of the expected response times of the child nodes to variations of the provisioned resources.

- If $\alpha \in (0, 1)$, then the two interpretations are combined, with higher or lower impact depending on the proximity to the limit values 0 and 1, respectively.

Note that, for each node composing actions through a fork-join construct, an optimal value of the hyperparameter α could be evaluated by exploiting numerical methods, though at the cost of increasing the computational complexity, which is here exacerbated by the model complexity and is undesirable for

reactive provisioners. An empirical evaluation to determine the optimal value of α is provided in Section VI-D.

E. Resource Accounting Heuristics

Repeated leaves are assigned different amounts of resources during the top-down split of the resource budget; in fact, all the repeated leaves of a set $h \in H$ of the RLT represent the same service in the system. To properly account for this, we assign resources to the entire set h through the function $\mathcal{P}(h) = \max_{v \in h} P(v)$. Then, in the bottom-up evaluation of the response time distribution, we split $\mathcal{P}(h)$ among the replicated leaves $v \in h$ according to one of the following heuristics. Formally, given an RLT $\langle V, E, v_0, H \rangle$, a provisioning $P : V \rightarrow \mathbb{R}_{>0}$ for each node $v \in V$, and a provisioning $\mathcal{P} : H \rightarrow \mathbb{R}_{>0}$ for each set $h \in H$ of replicated leaves, Algorithm 2 obtains a separate provisioning $P' : V \rightarrow \mathbb{R}_{>0}$ for each replicated leaf of $h \in H$ by splitting $\mathcal{P}(h)$ according to one of the following heuristics (lines 2-8):

- *Completely Agnostic Accounting (CAA)*: We split $\mathcal{P}(h)$ equally among the nodes of h (lines 3-4):

$$P'(v) = \frac{\mathcal{P}(h)}{|h|} \quad \forall v \in h. \quad (9)$$

- *Resource Proportional Accounting (RPA)*: We split $\mathcal{P}(h)$ proportionally to $P(v)$ (lines 5-6):

$$P'(v) = \frac{P(v)}{\sum_{u \in h} P(u)} \mathcal{P}(h) \quad \forall v \in h. \quad (10)$$

- *Effective Utilization Accounting (EUA)*: We account to each node v of h the fraction of $\mathcal{P}(h)$ representing the effective amount of resources consumed by v (lines 7-8):

$$P'(v) = \sum_{i=1}^{|G^h|} \frac{G_m^h}{|I_m^h|} \mathbf{1}_{I_m^h}(l) \quad \forall v \in h \quad (11)$$

where $G^h = \{P(v) \mid v \in h\}$ is the set of amounts assigned to the nodes of h , $I_m^h = \{u \in h \mid P(v) \geq G_m^h\}$ is the set of nodes of h for which $P(v)$ is not lower than the m -th element G_m^h of G^h , $|I_m^h|$ is the cardinality of I_m^h , and $\mathbf{1}_{I_m^h}(\cdot)$ is the indicator function over the set I_m^h .

For instance, suppose that our algorithm assigned $P(\text{B1}) = 2$, $P(\text{B2}) = 1$, and $P(\text{B3}) = 2$ in Fig. 3 and that the replicated set B is assigned $\mathcal{P}(\text{B}) = \max\{2, 1, 2\} = 2$. Then, CAA assigns $P'(\text{B1}) = P'(\text{B2}) = P'(\text{B3}) = 2/3$, and RPA assigns $P'(\text{B1}) = P'(\text{B3}) = (2 \cdot 2)/(2 + 1 + 2) = 4/5$ and $P'(\text{B2}) = (1 \cdot 2)/(2 + 1 + 2) = 2/5$. In EUA, $G^{\text{B}} = \{1, 2\}$: resources from 0 to 1 are divided among all the three nodes, while resources from 1 to 2 only between nodes B1 and B3. Thus, $P'(\text{B1}) = P'(\text{B3}) = 1/3 + 1/2$ and $P'(\text{B2}) = 1/3$.

F. Resource Provisioning for Services

Algorithm 4 shows our provisioning method. Specifically, given an RLT $\langle V, E, v_0, H \rangle$ with provisioning $P : V \rightarrow \mathbb{R}_{>0}$, first we balance resources assigned to the leaf nodes of the RLT through Algorithm 1 with total resource amount $P(v_0)$ (line 1), yielding a new provisioning $P' : V \leftarrow \mathbb{R}_{>0}$. Then, we compute

Algorithm 2: Evaluate Accounted Resources of the RLT.

```

AccountResources( $\langle V, E, v_0, H \rangle, a, \mathcal{P}, P$ )
input : RLT  $\langle V, E, v_0, H \rangle$ , accounting heuristics  $a$ ,
        replicated set provisioning  $\mathcal{P} : H \rightarrow \mathbb{R}_{>0}$ ,
        provisioning  $P : V \rightarrow \mathbb{R}_{>0}$ 
output : new provisioning  $P' : V \rightarrow \mathbb{R}_{>0}$ 
1 foreach node  $v$  do
2   if node  $v$  is a leaf and  $h \in H$  its replicated set then
3     if  $a$  is the CAA heuristics then
4       Compute  $P'(v)$  according to Eq. (9)
5     if  $a$  is the RPA heuristics then
6       Compute  $P'(v)$  according to Eq. (10)
7     if  $a$  is the EUA heuristics then
8       Compute  $P'(v)$  according to Eq. (11)
9   else
10     $P'(v) \leftarrow P(v)$ 
11  return  $P'$ 

```

the provisioning $\mathcal{P} : H \rightarrow \mathbb{R}_{>0}$ of each set $h \in H$ of replicated leaves as the maximum of the amounts assigned to the leaf nodes of h (lines 2-3), i.e., $\mathcal{P}(h) = \max_{v \in h} P(v)$, and we compute the accounted resources for each node of h by Algorithm 2 (line 4), obtaining the provisioning P' . Next, we compute the response time of each leaf node v with the new provisioning (line 5) by Algorithm 3, scaling provisioning from $P(v)$ to either $\mathcal{P}(h)$ (lines 2-3) or $P'(v)$ (lines 4-5) depending on whether v belongs to a replicated set h or not, respectively. In particular, scaling of provisioning is performed as discussed in Section V-B, e.g., for a shifted and truncated exponential random variable with support $[a, b]$ and rate λ , scaling provisioning from R_i to R_j yields a shifted truncated exponential random variable with support $[(R_i/R_j)a, (R_i/R_j)b]$ and rate $(R_j/R_i)\lambda$.

Then, we compute the resource amount R_{SLO} of (5) (line 6), i.e., the estimated minimum amount needed by our approach to fit the SLO, and we derive a new provisioning for each service by repeatedly performing the following steps until the SLO is satisfied or a maximum number of iterations is performed (lines 7-16): derivation of provisioning for the leaf nodes (line 10), and for each set h of replicated leaves as the sum of the amounts of the leaf nodes of h , i.e., $\mathcal{P}(h) = \sum_{v \in h} P(v)$ (lines 11-12); derivation of the accounted resources (line 13); evaluation of the response times of the leaf nodes (line 14), and, derivation of the workflow E2E response time CDF (line 15).

To derive the actual minimum amount of resources needed to fit the SLO, we perform a variant of Algorithm 4 where, at each iteration, we increase or decrease the amount R_{SLO} considered for the workflow by the provisioning method.

VI. EXPERIMENTAL METHODOLOGY

In this section, we describe the research objectives of the experiments (Section VI-A), the considered workflow models (Section VI-B), and the quantitative measures of the effectiveness of a resource provisioning (Section VI-C). Then, we select our hyperparameters (Section VI-D) and we illustrate alternative resource provisioning approaches (Section VI-E), the considered relations between service durations and provisioned

Algorithm 3: Evaluate Response Times of Elementary Services by Scaling Their Resource Provisioning.

```

EvaluateRespTimes( $\langle V, E, v_0, H \rangle, \mathcal{P}, P, P', \tau$ )
  input : RLT  $\langle V, E, v_0, H \rangle$ ,
          replicated set provisioning  $\mathcal{P} : H \rightarrow \mathbb{R}_{>0}$ ,
          previous provisioning  $P : V \rightarrow \mathbb{R}_{>0}$ ,
          current provisioning  $P' : V \rightarrow \mathbb{R}_{>0}$ , response
          time random variables  $\tau(v), v \in L$  of the nodes
          with provisioning  $P$ 
  output : response time  $\tau'(v), v \in L$  with provisioning  $P'$ 
1 foreach leaf node  $v$  do
2   if node  $v$  belongs to a replicated set  $h$  then
3     Evaluate the response time  $\tau'(v)$  of  $v$  from  $\tau(v)$ 
4     by scaling provisioning from  $P(v)$  to  $\mathcal{P}(h)$ 
5   else
6     Evaluate the response time  $\tau'(v)$  of  $v$  from  $\tau(v)$ 
7     by scaling provisioning from  $P(v)$  to  $P'(v)$ 
8 return  $\tau'$ 
  
```

Algorithm 4: Evaluate Resource Provisioning for Services.

```

ComputeProvisioning( $\langle V, E, v_0, H \rangle, a, X_{\text{SLO}}, \epsilon, P, \tau, I$ )
  input : RLT  $\langle V, E, v_0, H \rangle$ , accounting heuristics  $a$ ,
          SLO  $X_{\text{SLO}}$ , SLO compliance threshold  $\epsilon$ ,
          provisioning  $P : V \rightarrow \mathbb{R}_{>0}$ , response times
           $\tau(v), v \in L$  with provisioning  $P$ ,
          maximum number  $I$  of iterations
  output : new provisioning  $P' : V \rightarrow \mathbb{R}_{>0}$ 
1  $P' \leftarrow \text{VisitRLT}(\langle V, E, v_0, H \rangle, P, P(v_0), \tau)$ 
2 foreach  $h \in H$  do
3    $\mathcal{P}(h) \leftarrow \max_{v \in h} P(v)$ 
4  $P' \leftarrow \text{AccountResources}(\langle V, E, v_0, H \rangle, a, \mathcal{P}, P')$ 
5  $\tau \leftarrow \text{EvaluateRespTimes}(\langle V, E, v_0, H \rangle, \mathcal{P}, P, P')$ 
6 Compute  $R_{\text{SLO}}$  by Eq. (5)
7 iterations  $\leftarrow 0$ 
8 do
9    $P \leftarrow P'$ 
10   $P' \leftarrow \text{VisitRLT}(\langle V, E, v_0, H \rangle, P, R_{\text{SLO}}, \tau)$ 
11  foreach  $h \in H$  do
12     $\mathcal{P}(h) \leftarrow \sum_{v \in h} P(v)$ 
13   $P' \leftarrow \text{AccountResources}(\langle V, E, v_0, H \rangle, a, \mathcal{P}, P)$ 
14   $\tau \leftarrow$ 
15    EvaluateRespTimes( $\langle V, E, v_0, H \rangle, \mathcal{P}, P, P'$ )
16    Derive CDF of the workflow response time  $X$  by [62]
17    iterations  $\leftarrow$  iterations + 1
18 while  $|\delta(X, X_{\text{SLO}}) - \frac{1}{2}| > \epsilon \wedge \text{iterations} \leq I$ ;
19 return  $P'$ 
  
```

resources (Section VI-F), and a simulator of the execution of workflow requests at high workloads (Section VI-G).

A. Aim and Scope

We assess feasibility and effectiveness of our approach by a thorough experimentation, considering both scenarios where the assumptions of our surrogate performance model hold (i.e., low workload and service durations scaling with resources according to a homogeneous linear relation) and scenarios where they are violated. Moreover, our approach is compared with alternative heuristics for fitting an SLO on the workflow E2E response time CDF while minimizing provisioned resources, considering workflow topologies derived from real benchmark applications

or randomly generated with controlled statistics. Specifically, we consider the following research questions:

- *RQ1*: In a low-workload scenario where service durations scale with resources according to a homogeneous linear relation, does our approach outperform alternative baselines (with more limited information) in fitting the SLO using a given resource amount?
- *RQ2*: In a low-workload scenario where service durations scale with resources according to a homogeneous linear relation, does our approach outperform baselines in minimizing provisioned resources while fitting the SLO?
- *RQ3*: In scenarios with high workload or where service durations do not scale with resources according to a homogeneous linear relation, does our approach outperform baselines in the terms defined by RQ1 and RQ2?
- *RQ4*: Is the approach feasible for workflows with topology derived from real benchmark applications?
- *RQ5*: Is the approach feasible for workflows with topology composing a very high number of services with sequence, fork-join, split-merge, and irreducible DAG constructs?

B. Workflow Models

1) *Benchmark Topologies*: Fig. 4 shows workflow topologies derived from widely used real-world benchmarks. Specifically:

- Fig. 4(a) shows a well-nested topology defined in [63] for a microservices application, consisting of 11 elementary actions composed by sequence and split-merge constructs.
- Fig. 4(b) shows the topology of a workflow reported in [64] for the Social Network application of DeathStar-Bench [65], an open-source benchmark suite for cloud microservices. The topology is well-nested and consists of 7 elementary actions composed by sequence and fork-join constructs.
- Fig. 4(c) shows the topology of the Epigenomics workflow of the Pegasus Workflow Repository [66], [67]. The topology is well-nested and consists of 20 elementary actions composed by sequence and fork-join constructs.
- Fig. 4(d) shows the topology of the Montage workflow of the Pegasus Workflow Repository [66], [67]. The topology is not-well-nested and consists of 25 elementary actions composed by sequence and irreducible DAG constructs.

2) *Randomly Generated Topologies*: We randomly generate a suite of non-well-nested workflow topologies. To this end, we generate, as top-level action of the activity diagram (e.g., action TOP in Fig. 2), an irreducible DAG with a number of child actions between 4 and 9. The remaining actions are built as a composition of child actions by sequence, fork-join, and split-merge operators, randomly selected with probabilities 0.4, 0.4, and 0.2, respectively, considering two complexity factors: *i*) depth D , i.e., the maximum number of nested actions of the activity diagram, excluding the top-level one; and, *ii*) breadth B , i.e., the maximum number of children of an action of the activity diagram, excluding the top-level one. We generate 20 workflow topologies for each combination of values of $D, B \in \{2, 3, 4\}$,

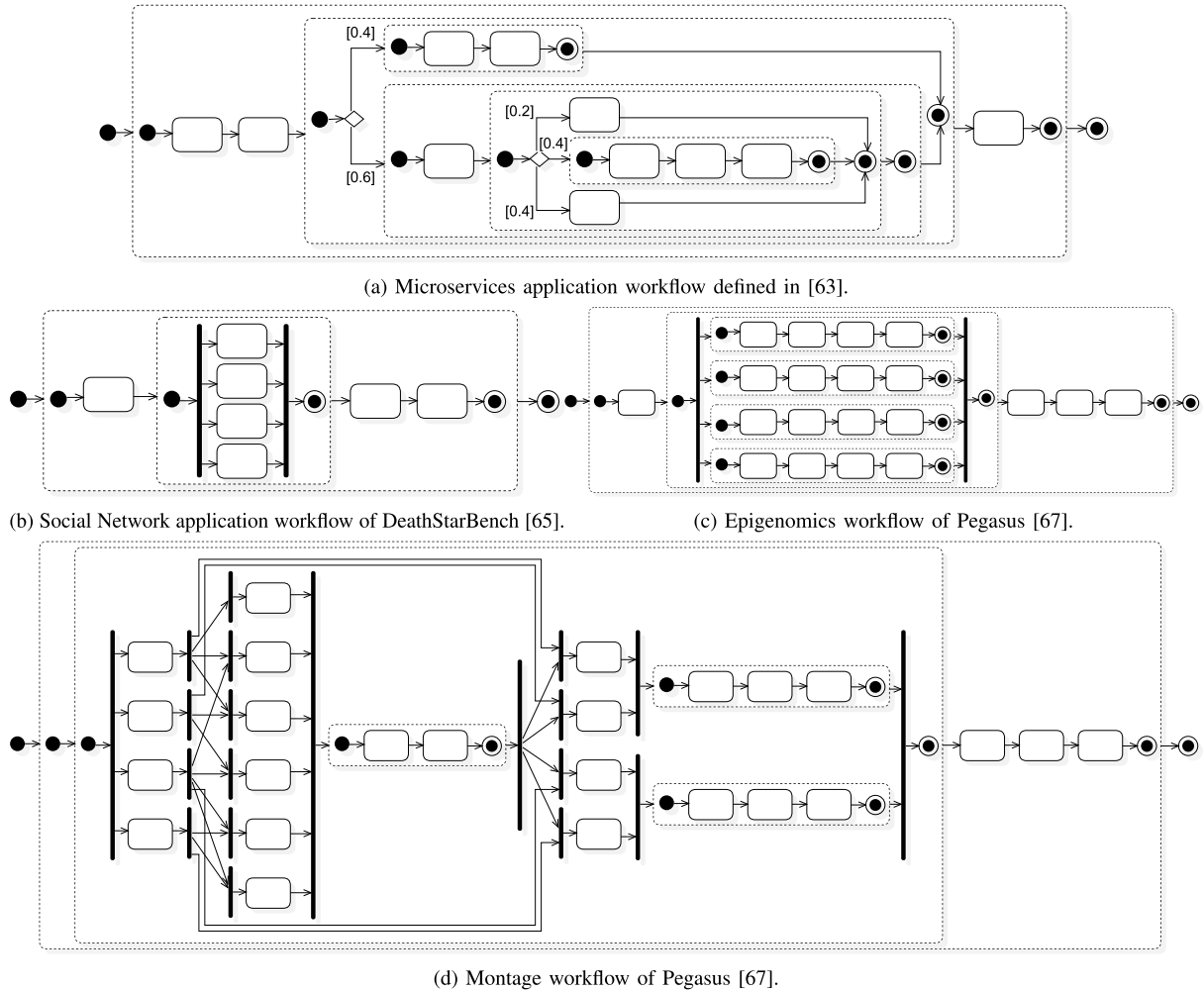


Fig. 4. Workflow topologies obtained from widely used real-world benchmarks.

yielding 180 different topologies with a number of actions comprised between 10 and 704.

3) *Service Execution Time Distributions*: For each topology, we define the job size of each elementary service by assigning a unitary amount of resources to the service and using the execution time distribution obtained by approximating a histogram of samples derived from the WS-Dream dataset [60], which collects response times of many web services. For each of 100 considered web services, we derive a histogram by collecting and regularizing the related samples according to the inter-quartile range rule [68]. Then, for each elementary service of each workflow, we randomly select a histogram and we approximate it with a shifted truncated exponential CDF fitting its mean value and its coefficient of variation [59].

4) *Workflow SLO*: For each workflow, we define an SLO on the duration CDF of each elementary service as a 5-phase Erlang CDF, with rate λ derived from the expected service duration μ : with probability 0.5, $\lambda = 5/\mu$; with probability 0.5, $\lambda = (1 + 0.5 \cdot s \cdot r)/\mu$ where s is randomly drawn in $\{-1, +1\}$, and r is a rational number randomly drawn in $[-1, 1]$. Thus, in half of the cases the expected value of the service SLO is equal to μ , and in the other half it is lower (i.e., under-provisioning) or larger (i.e., over-provisioning) than μ . Then, we define the

workflow SLO as the E2E response time CDF of the workflow under low workload with each elementary service distributed as its individual SLO, and we evaluate it by the method of [62]. In doing so, we consider cases of workflow under-provisioning and cases of workflow over-provisioning.

C. Quantitative Measures of Interest

For a workflow with response time X (for some provisioning to its elementary services), we measure its compliance with an SLO X_{SLO} through the pairwise comparison dominance $\delta(X, X_{SLO})$, which provides an average measure of how many times X anticipates X_{SLO} . To quantify the anticipation, we derive the time improvement $\iota(X, X_{SLO})$ of X with respect to X_{SLO} :

$$\iota(X, X_{SLO}) := \int F(x) - \min(F(x), F_{SLO}(x)) dx \quad (12)$$

where F and F_{SLO} are the CDFs of X and X_{SLO} , respectively. Note that $\iota(X, X_{SLO})$ only considers when F anticipates F_{SLO} , avoiding penalization when F is late with respect to F_{SLO} .

Moreover, $\iota(X, X_{\text{SLO}})$ can be lower bounded by

$$\beta(X, X_{\text{SLO}}) := \frac{E[X] - E[X_{\text{SLO}}]}{E[X]} \quad (13)$$

where $E[X]$ and $E[X_{\text{SLO}}]$ denote the expected values of X and X_{SLO} , respectively. In fact, by definition of $\iota(X, X_{\text{SLO}})$, $\iota(X, X_{\text{SLO}}) \geq \int (F(x) - F_{\text{SLO}}(x)) dx = E[X] - E[X_{\text{SLO}}]$. Moreover, we can guarantee that $E[X] \geq 1$ by changing the unit measure of time (e.g., converting minutes to seconds). Hence, $E[X] - E[X_{\text{SLO}}] \geq (E[X] - E[X_{\text{SLO}}])/E[X]$, which finally proves that $\iota(X, X_{\text{SLO}}) \geq \beta(X, X_{\text{SLO}})$.

D. Selection of Hyperparameters

We investigate how the hyperparameter α (see Section V-D) and the resource accounting heuristics (see Section V-E) impact the capacity of our approach to improve the workflow E2E response time by balancing the resources assigned to its elementary services. We randomly generate 200 workflow topologies for each combination of values of the workflow depth $D \in \{2, 3, 4\}$ and breadth $B \in \{2, 3, 4\}$ as described in Section VI-B2, yielding 1800 different topologies with a number of actions comprised between 16 and 2304. For each topology, we associate each elementary service with an execution time distribution derived as illustrated in Section VI-B3.

For each model, each value of $\alpha \in \{0, \frac{1}{8}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}, 1\}$, and each account heuristics (i.e., CAA, RPA, and EUA), we derive the pairwise-comparison dominance $\delta(X, X_{\text{init}})$ between the initial workflow E2E response time X_{init} and the E2E response time X obtained after balancing the initial amount of provisioned resources according to our approach. For each accounting heuristic, results show that $\frac{1}{4}$ is the best value of α , that is $\alpha = \frac{1}{4}$ yields the lowest value of $|\delta(X, X_{\text{init}}) - \frac{1}{2}|$ in the majority of the cases, i.e., more than 50% of the cases. The number of cases with $\alpha \in \{\frac{1}{8}, \frac{3}{8}\}$ being the best choice increases with D and B . Specifically, $\alpha \in \{\frac{1}{4}, \frac{3}{8}\}$ is the best choice in more than 70% of the cases, and $\alpha \in \{\frac{1}{8}, \frac{1}{4}, \frac{3}{8}\}$ in more than 85% of the cases. Conversely, the impact of the accounting heuristics appears insensitive to the choice of α .

To confirm such observations, we execute 4 hypothesis tests:

- The first test evaluates if the choice of the heuristic is statistically relevant in determining the best value of α . We perform the Kruskal-Wallis test [69], where the null hypothesis is “*The accounting heuristics are not statistically relevant in determining the best α value*”, and the significance level is 0.05. For all the 1800 generated workflows, the test returns a p -value near 1.00, meaning that the null hypothesis cannot be rejected, and thus that there is no statistical relevance connecting the choice of the best α value with the accounting heuristics.
- The remaining three tests evaluate whether $\alpha \in \{\frac{1}{4}, \frac{3}{8}, \frac{1}{8}\}$ is the best choice with statistical relevance. We consider the following three null hypotheses: “ $\alpha = \frac{1}{4}$ is the best choice for at least 50% of the cases”; “ $\alpha \in \{\frac{1}{4}, \frac{3}{8}\}$ is the best choice for at least 70% of the cases”; and, “ $\alpha \in \{\frac{1}{8}, \frac{1}{4}, \frac{3}{8}\}$ is the best choice for at least 85% of the cases”. For each hypothesis, we perform an upper-tail binomial test [70]

with significance level 0.05. In all the cases, we obtain p -value lower than 0.05, thus rejecting the null hypotheses and confirming that the values $\alpha = \frac{1}{4}$, $\alpha \in \{\frac{1}{4}, \frac{3}{8}\}$, and $\alpha \in \{\frac{1}{8}, \frac{1}{4}, \frac{3}{8}\}$ are the best choice with statistical significance or more than 50%, 70% and 85% of the cases, respectively. This result suggests that it is preferable to select a value of α that tends to balance both the response times of elementary service and their sensitivity to variations of provisioned resources.

Then, we investigate how effective $\alpha = \frac{1}{4}$ is in reducing the E2E response time of a workflow with respect to the actual best value of α for that workflow. To this end, for each workflow and each accounting heuristics, we compute the pairwise-comparison dominance $\delta(X_{\alpha=\frac{1}{4}}, X_{\alpha=\text{best}})$ between the workflow E2E response times $X_{\alpha=\frac{1}{4}}$ and $X_{\alpha=\text{best}}$ achieved by balancing provisioned resources by our approach with $\alpha = \frac{1}{4}$ and with the best value of α for that workflow, respectively. We also compute the corresponding lower bound $\beta(X_{\alpha=\frac{1}{4}}, X_{\alpha=\text{best}})$ on the relative improvement of the E2E response time. Results show that $\delta(X_{\alpha=\frac{1}{4}}, X_{\alpha=\text{best}}) \in [0.46, 0.50]$ for each accounting heuristics, very close to the equivalence value of 0.5, proving that $\alpha = \frac{1}{4}$ is still effective even when it is not the best choice. This fact is also evident from the values of $\beta(X_{\alpha=\frac{1}{4}}, X_{\alpha=\text{best}})$, which are almost equal to 0. According to this, $\alpha = \frac{1}{4}$ and the EUA accounting heuristics are selected.

These experiments highlight the complexity of the problem. Values of pairwise-comparison dominance close to 0.5 suggest that different provisioning yielded by different α values produce similar results in terms of E2E response time, indicating that more provisioning solutions optimize the E2E response time. Determining such solutions is not trivial due to the many different factors that could be considered in the selection.

E. Baseline Resource Provisioning Techniques

We define alternative resource provisioning methods ignoring some prerequisite information of our approach, performing an ablation study aimed at determining which prior knowledge is more significant for our purpose, i.e., fitting the workflow SLO while minimizing provisioned resources. Specifically:

- *Completely Agnostic Provisioning (CAP)*: It assigns the same amount of resources to each elementary service, ignoring the workflow topology, the duration of elementary services and their sensitivity to the scaling of provisioned resources. Given a total amount R of resources and a workflow made of elementary services a_1, \dots, a_N , the resource provisioning is $P(a_i) = R/N \forall i \in \{1, \dots, N\}$.
- *Topology Driven Provisioning (TDP)*: It allocates resources to elementary services by considering the workflow topology. Given a total amount R of resources and a workflow made of elementary services a_1, \dots, a_N , the resource allocation is obtained by applying our approach with $\alpha = \frac{1}{4}$ assuming that each service a_i has deterministic duration equal to its mean value $\mu_i \forall i \in \{1, \dots, N\}$.
- *Unbalanced Duration Driven Provisioning (UDDP)*: It provides each elementary service with an amount of resources proportional to its expected duration, ignoring the

workflow topology and the sensitivity of service durations to resource scaling. Given a total amount R of resources and a workflow made of elementary services a_1, \dots, a_N with mean duration μ_1, \dots, μ_N , respectively, the resource provisioning is $P(a_i) = R\mu_i / \sum_j \mu_j \forall i \in \{1, \dots, N\}$.

- **Balanced Duration Driven Provisioning (BDDP):** It allocates resources to elementary services with the aim of balancing their expected execution times. Given a total amount R of resources and a workflow made of elementary services a_1, \dots, a_N , the resource allocation is obtained by applying our approach with $\alpha = 0$ to a composition of a_1, \dots, a_N by a fork-join construct.
- **Sensitivity Driven Provisioning (SDP):** It allocates resources to elementary services with the aim of balancing their sensitivity to the scaling of provisioned resources, thus assigning more resources to less sensitive activities. Given a total amount R of resources and a workflow made of elementary services a_1, \dots, a_N , the resource allocation is obtained by applying our approach with $\alpha = 1$ to a composition of a_1, \dots, a_N by a fork-join construct.

F. Other Models of Service Performance

We experiment with other models of response time scaling to validate our approach in scenarios where the homogeneous linear relation of (3) is violated. Given a service provisioned with resource amount R , we consider the following models of service response time $T(R)$:

- **Piecewise Linear:** The response time $T(R)$ scales linearly with provisioned resources R up to a maximum value R_{\max} , and then it remains constant, modeling the fact that, as provisioning increases, the response time of the non-parallelizable fraction of the job size prevails over the response time of the parallelizable fraction:

$$T(R) := \frac{J}{\min\{R, R_{\max}\}}. \quad (14)$$

- **Inhomogeneous Linear:** The response time $T(R)$ scales with provisioned resources R according to a inhomogeneous linear relation representing the Amdahl's Law [71]:

$$T(R) := \left(\frac{p}{R} + (1 - p) \right) J \quad (15)$$

where p is the fraction of the job size J that would benefit from the improvement of provisioned resources, and $1 - p$ is the job size fraction that would not benefit from it (e.g., only the parallel portion of a program benefits from additional CPU cores).

For a service with job size J modeled as a shifted and truncated Exponential CDF with support $[a, b]$ and rate λ , the response time $T(R)$ for a given amount of resources R is also distributed as a shifted and truncated Exponential CDF where:

- If (14) is used, $T(R)$ has support $[a/\eta, b/\eta]$ and rate $\lambda\eta$ where $\eta = \min\{R, R_{\max}\}$.
- If (15) is used, $T(R)$ has support $[a/\beta, b/\beta]$ and rate $\lambda\beta$ where $\beta = 1/(p/R + 1 - p)$.

G. High-Workload Scenario Simulator

We implemented a simulator of the execution of workflow requests, which considers queue effects and performs horizontal scaling actions using the resources available after the initial provisioning. We model the arrival of workflow requests as a Poisson process (i.e., with exponential interarrival times); at each arrival, we sample an instance of the workflow, including the selection of activities in split-merge (XOR) nodes and the sampling of a job size for each activity. Activities of the workflow are served by dedicated multiserver queues with FIFO policies: initially, for each initial activity of the workflow (i.e., without dependencies), a job is added to the multiserver queue allocated for its execution; as soon as all the dependencies of an activity of the workflow are satisfied (i.e., their jobs completed), a job is added to the multiserver queue associated with the activity. Periodically, the simulator checks the utilization of each multiserver queue: if it is higher than a threshold c for N consecutive checks, another server is added to serve the queue, as long as there are sufficient remaining resources. Then, utilization of the queue is not checked for M periods (i.e., the ‘‘cool down period’’).

VII. EXPERIMENTAL RESULTS

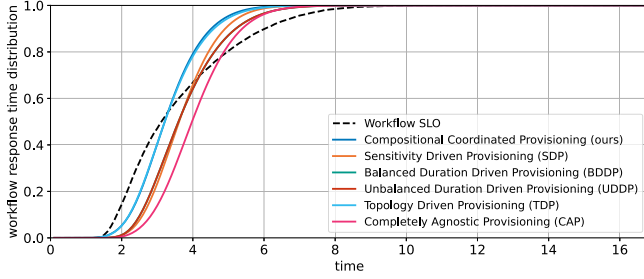
In this section, we present the results obtained with a low workload (Section VII-A) and a high workload (Section VII-B) of workflow requests, considering the cases that: *i*) all provisioning methods use the actual minimum resources needed by our approach to fit the SLO (**fixed-res case**); and *ii*) each provisioning method uses its actual minimum resources needed to fit the SLO (**custom-res case**). Resources used in the fixed-res and custom-res cases are the same for our approach, while they may be different for alternative approaches. In fact, each approach balances resources differently (line 1 of Algorithm 4), yielding a different amount R_{SLO} (line 6 of Algorithm 4).

In the fixed-res and custom-res cases with high workload, horizontal scaling is disabled to observe the impact of workflow requests on the response time. We also consider a variant of the custom-res case with horizontal scaling enabled with (virtually) unlimited amount of resources available (**custom-res-hs case**).

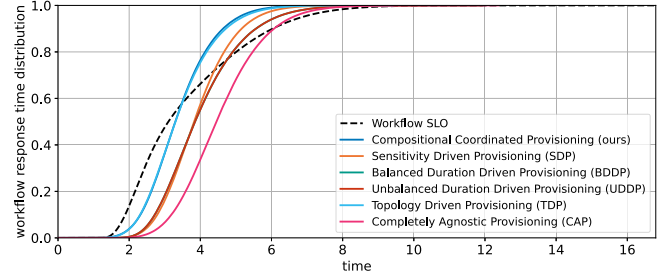
In each experiment, we compute the CDF of the workflow E2E response time achieved by each provisioning approach, measuring the accuracy in fitting the SLO in terms of dominance deviation (i.e., $|\delta(X, X_{\text{SLO}}) - \frac{1}{2}|$) and time improvement. In Algorithm 4, we consider the maximum number of iterations I equal to 20 for the 4 benchmark models and equal to 5 for the 180 models with randomly generated topology. For the piece-wise linear performance model, we use $R_{\max} = 1$; for the inhomogeneous linear performance model, we use $p = 0.7$.

A. Low-Workload Scenario

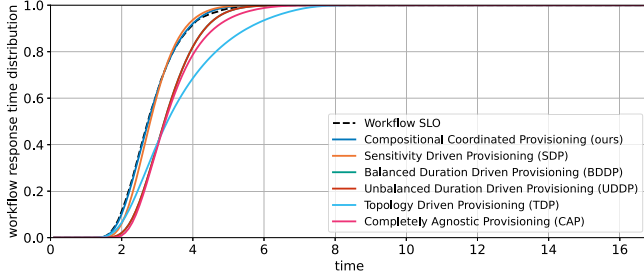
1) **Homogeneous Linear Performance Model:** For the **fixed-res case**, Fig. 5(a) plots the results for the workflow of Fig. 4(a). Our approach obtains the smallest dominance deviation (0.001), slightly lower than that obtained by TDP (0.004) and nearly two orders of magnitude lower than those obtained by the other methods (ranging from 0.085 for SDP to 0.145 for CAP). Fig. 5(c)



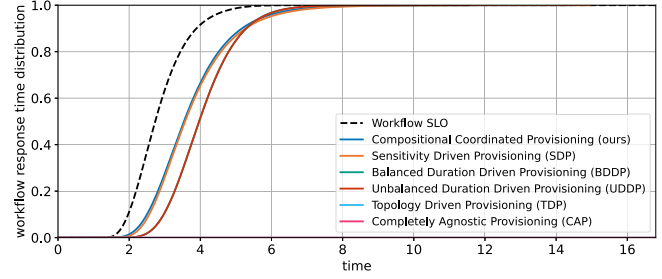
(a) Microservices application workflow defined in [63].



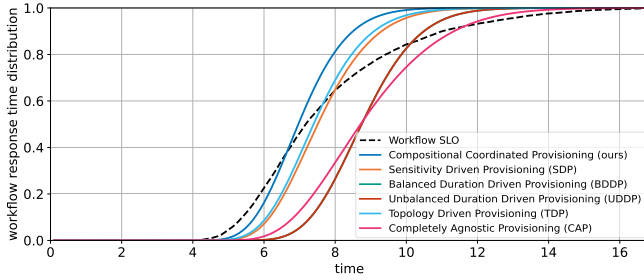
(b) Microservices application workflow defined in [63].



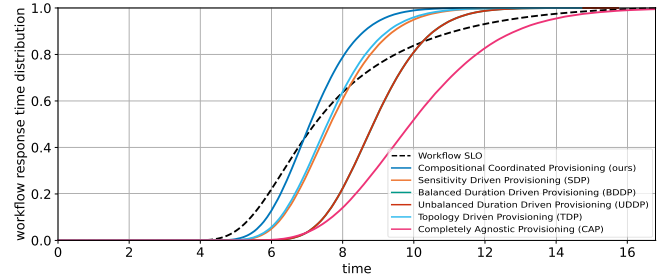
(c) Social Network application workflow of DeathStarBench [65].



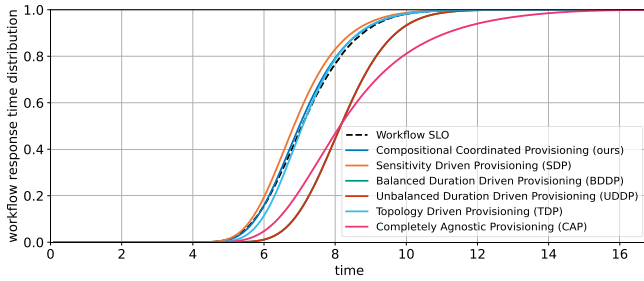
(d) Social Network application workflow of DeathStarBench [65].



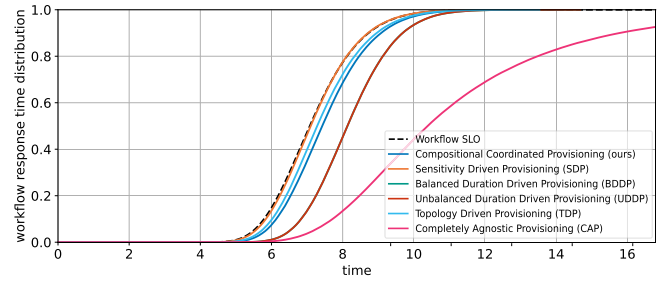
(e) Epigenomics workflow of Pegasus [67].



(f) Epigenomics workflow of Pegasus [67].



(g) Montage workflow of Pegasus [67].



(h) Montage workflow of Pegasus [67].

Fig. 5. Fixed-res case with homogeneous linear performance model: low workload (a-c-e-g) and high workload (b-d-f-h).

shows the results for the workflow of Fig. 4(b), for which our approach again achieves the smallest dominance deviation (0.003), one order of magnitude lower than that achieved by SDP (0.026) and two orders of magnitude lower than those obtained by the other methods (ranging from 0.161 for TDP to 0.182 for CAP). Fig. 5(e) plots the results for the workflow of Fig. 4(c), for which our approach obtains the third smallest dominance deviation (0.069) after TD (0.009) and SD (0.029). Fig. 5(g) plots the results for the workflows of Fig. 4(d), for which our approach obtains the second smallest dominance deviation (0.028) after TD (0.004). For the time improvement, our method obtains the

largest value in Fig. 5(a) (0.444) and in Fig. 5(e) (0.837), and the second largest value in Fig. 5(c) (0.013) after SD (0.033) and in Fig. 5(g) (0.056) after SD (0.211), confirming the results of our approach as positive.

For workflows with randomly generated topology, i.e., 20 models for each pair (D, B) of values of depth $D \in \{2, 3, 4\}$ and breadth $B \in \{2, 3, 4\}$, our method yields the smallest dominance deviation for 35% of the models with (2,2), 70% of the models with (2,3), 75% of the models with (4,4), 85% of the models with (3,2), 95% of the models with (4,3), and 100% of the models with (2,4), (3,3), (3,4), (4,2). Thus, the accuracy

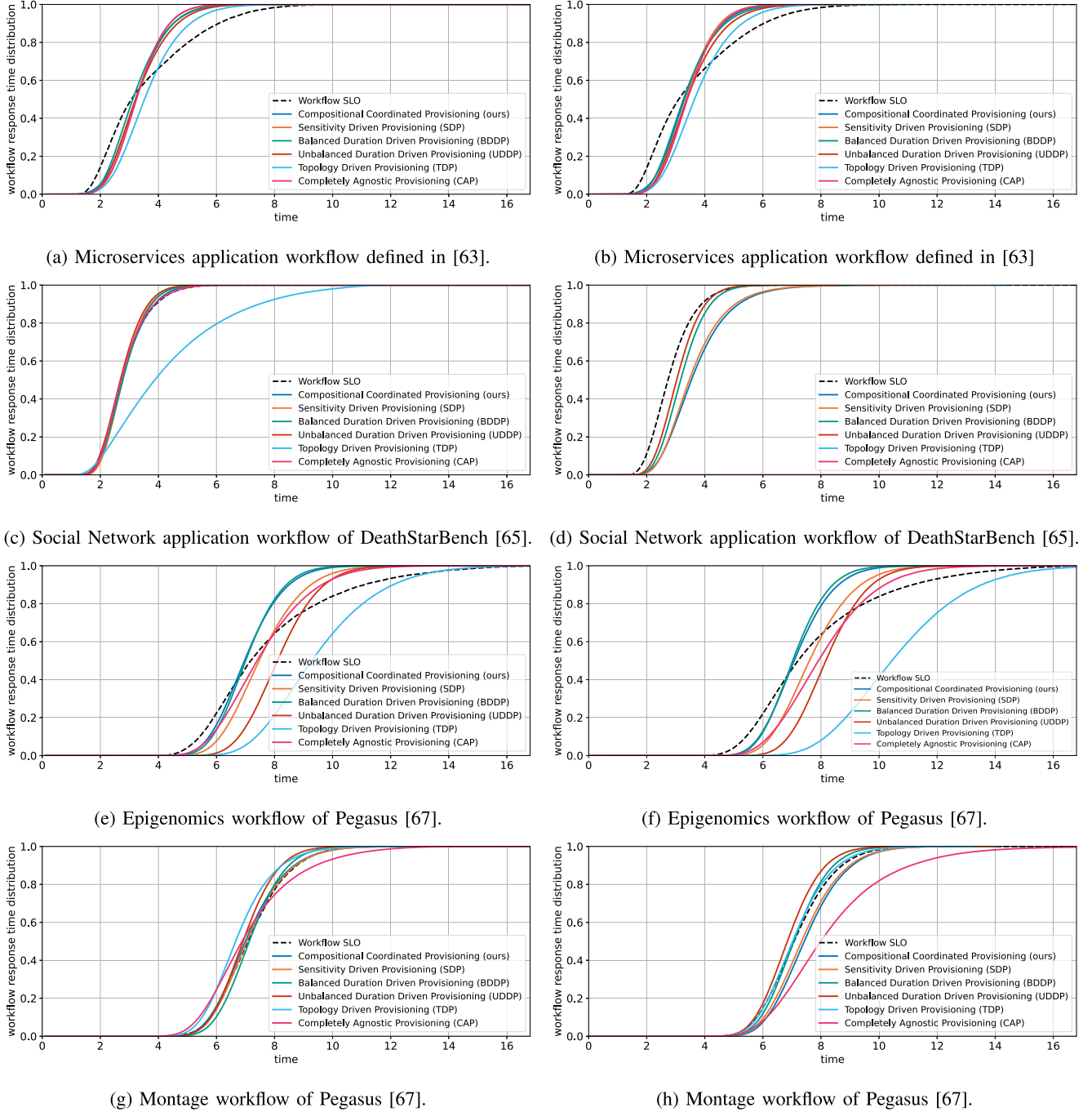


Fig. 6. Custom-res case with homogeneous linear performance model: low workload (a-c-e-g) and high workload (b-d-f-h).

of our method increases with model complexity in terms of number of services and topology of their composition, which is expected given that simpler models, with fewer services to optimize, reduce the improvement margin of our approach.

For the **custom-res** case, Fig. 6(a), (c), (e), (g) plot the results for the workflows of Fig. 4(a)–(d), respectively. In Fig. 6(a), we use the second smallest resources (7.8) after TDP (7.3) while providing the second smallest dominance deviation (0.038) after UDDP (0.010). In Fig. 6(c), we nearly use the smallest resources (6.2), obtained by SDP, while achieving the smallest dominance deviation (0.003), equal to the one of CAP and one order of

magnitude lower than the one of SDP (0.012). In Fig. 6(e), we use the second smallest resources (13.3), slightly larger than those of SDP (13.2), while providing the third smallest dominance deviation (0.075) after CAP (0.016 against 16.8 resources) and BDDP (0.074 against 16.4 resources). In Fig. 6(g), we use the smallest resources (25.0) while achieving the third smallest dominance deviation (0.054) after SDP (0.015 against 25.2 resources) and CAP (0.042, against 30.7 resources). Overall, our method yields the best tradeoff between used resources and SLO compliance, even for workflows with very complex topology as in Fig. 4(d).

For workflows with randomly generated topology for each pair (D, B) , we use the smallest resources for 35% of the models with (2,2), 65% of the models with (2,3), 90% of the models with (3,2), 95% of the models with (2,4), and 100% of the models with (3,3), (3,4), (4,2), (4,3), (4,4), with significant savings (e.g., for (4,4), we use 232.0 resources on average, against 360.3, 388.5, 469.2, 469.6, and 550.6 used by SDP, BDDP, UDDP, CAP, and TDP, respectively). Moreover, for each pair (D, B) , while the percentage of models for which we provide the smallest dominance deviation decreases with respect to the fixed-res case, we obtain a value in the order of magnitude of the smallest one for the large majority of models.

2) *Other Performance Models*: For the **fixed-res** case (i.e., same provisioning for each service as in the fixed-res case with homogeneous linear performance model), with piece-wise and inhomogeneous linear performance models we obtain the smallest dominance deviation or a value in the same order of magnitude of the smallest one for the workflows of Fig. 4, except for the one in Fig. 4(c) with the inhomogeneous linear performance model, for which we obtain deviation 0.120, larger than the smallest one (0.034) obtained by SDP.

For the workflows with randomly generated topology, for each pair (D, B) , we obtain the smallest dominance deviation for a lower number of workflows (especially with the inhomogeneous linear model) though still for the relative majority (except for (2,2) and (2,3) with the piecewise linear model, for which we obtain the smallest deviation for 20% and 25% of the workflows, respectively, though mostly in the same order of magnitude of the smallest one for the remaining workflows).

For the **custom-res** case (i.e., same provisioning for each service as in the custom-res case with homogeneous linear performance model), for the 4 workflows of Fig. 4, the results in terms of dominance deviation remain similar to those with the homogeneous linear performance model. For the 180 workflows with randomly generated topology, the percentage of cases for which our method yields the smallest dominance deviation decreases, especially with the piecewise performance model, though values quite remain in the same order of magnitude.

3) *Remarks*: The results answer RQ1 and RQ2, pointing out that, in a low-workload scenario where service durations scale with resources according to a homogeneous linear relation, our approach outperforms baselines in fitting the SLO with better accuracy while using the same resources, and in minimizing the resources used to fit the SLO. The results also show that these claims quite hold also for the other performance models, thus positively answering RQ3 for low-workload scenarios.

B. High-Workload Scenario

For the fixed-res and custom-res cases, we have exponential inter-arrival times with rate equal to $\frac{2}{5}\eta$ for the 4 benchmark workflows and equal to $\frac{4}{5}\eta$ for the 180 workflows with randomly generated topology (where η is the expected value of the workflow SLO), which is sufficient to observe queue effects while maintaining the queues stable for the majority of the workflows. For the custom-res-hs case, with the horizontal scaler enabled, we consider a larger workload with rate equal to $\frac{1}{3}\eta$. Moreover, in

each experiment, we use $c = 0.7$ (utilization threshold), $N = 1$ (number of consecutive utilization checks before scaling), and $M = 1$ (number of cool down periods).

1) *Homogeneous Linear Performance Model*: For the **fixed-res** case (i.e., same provisioning for each service as in the fixed-res case with low workload), Fig. 5(b), (d), (f), and (h) plot the results for the workflows of Fig. 4(a)–(d), respectively, showing an increase in the response time (i.e., a shift to the right of the response time CDF) of each approach with respect to the low-workload scenario of Fig. 5(a), (c), (e), and (g), respectively (the increase is more or less marked for each approach). Notably, we obtain the smallest dominance deviation in Fig. 5(b) (0.021), Fig. 5(d) (0.25), and Fig. 5(f) (0.038), and the third smallest one (0.086) in Fig. (h) after SDP (0.005) and TDP (0.053).

For workflows with randomly generated topology for each pair (D, B) , we obtain similar results as in the low-workload scenario, notably with accuracy increasing with model complexity, proving the effectiveness of our resource allocation. Specifically, we obtain the smallest dominance deviation for 45% of the models with (2,2), 80% of the models with (3,2), 85% of the models with (2,4), 90% of the models with (2,3), (4,2), (4,4), and 100% of the models with (3,3), (3,4), (4,3).

For the **custom-res** case (i.e., same provisioning for each service as in the custom-res case with low workload), Fig. 6(b), (d), (f), and (h) plot the results for the workflows of Fig. 4(a)–(d), respectively, showing an increase in the response time (i.e., a shift to the right of the response time CDF) of each approach with respect to the low-workload scenario of Fig. 6(a), (c), (e) and (g). In Fig. 6(b), we obtain the second smallest dominance deviation (0.021) after BDD (0.020), while using the second smallest resources (7.8) after TDP (7.3). In Fig. 6(d), we obtain the fourth smallest dominance deviation (0.247) after UDDP (0.111), BDDP (0.158), and SDP (0.238), while using the smallest resources (6.2). In Fig. 6(f), we obtain the smallest dominance deviation (0.038) while using the second smallest resources (13.3) after SDP (13.2). In Fig. 6(h), we obtain the fifth smallest dominance deviation (0.085) after BDDP (0.006), TDP (0.013), UDDP (0.066), SDP (0.063) while using the smallest resources (25.0). Except for the latter model, we obtain the best tradeoff between used resources and SLO compliance.

For workflows with randomly generated topology, the percentage of models for which our method provides the smallest dominance deviation slightly decreases with respect to the custom-res case of the low-workload scenario.

For the **custom-res-hs** case, our capacity of minimizing the dominance deviation is no longer evident, as horizontal scaling compensates for inefficiencies in any provisioning strategy by dynamically deploying additional instances to replicate critical services when required. While this flexibility incurs significant resource costs due to replica proliferation, our solution emerges as the optimal provisioning method for resource savings. Specifically, we use the smallest resources for the workflows of Fig. 4(a) and (d), and the second smallest resources for the workflow of Fig. 4(b) (32.6) after UDDP (31.2) and for the workflow of Fig. 4(c) (55.9) after CAP (54.1). For workflows with randomly generated topology for each pair (D, B) , we use the smallest resources for 45% of the models with (2,2), 50% of the models with

(2,3), 80% of the models with (2,4), (3,2), 95% of the models with (3,3), (4,2), and 100% of the models with (3,4), (4,3), (4,4). As expected, our advantage increases with model complexity. Notably, average resource savings exceed 14% for the models with (3,4), (4,3), and (4,4), with peak benefits for the models with (4,4), i.e., 24.73% reduction in average resource consumption compared to SDP, which is the second-best resource-efficient choice in this case.

2) *Other Performance Models*: For the **fixed-res** case (i.e., same provisioning for each service as in the fixed-res case with homogeneous linear performance model), with piece-wise and inhomogeneous linear performance models we obtain the smallest dominance deviation or a value in the same order of magnitude of the smallest one for the models of Fig. 4.

For the workflows with randomly generated topology, for each pair (D, B) , we obtain the smallest dominance deviation for a lower number of workflows (especially with the inhomogeneous linear model) though still for the relative majority (except for (2,2) and (3,2) with the piecewise linear model, for which we obtain the smallest deviation for 15% and 20% of the workflows, respectively, though mostly in the same order of magnitude of the smallest one for the remaining workflows).

For the **custom-res** case (i.e., same provisioning for each service as in the custom-res case with homogeneous linear performance model), our method performs worse in minimizing the dominance deviation, especially with the piece-wise linear performance model, though obtaining dominance deviation in the order of magnitude of the smallest one in most of the cases.

For the **custom-res-hs** case (i.e., with horizontal scaling), our method performs worse in minimizing resources, especially with the piece-wise linear performance model, though it remains the best option for the large majority of complex models.

3) *Remarks*: The results answer RQ3, pointing out that, in high-workload scenarios where service durations scale with resources according to a homogeneous linear relation, our method outperforms baselines in fitting the SLO with better accuracy using the same resources, and in minimizing the resources needed to fit the SLO. These claims quite hold also for the other performance models. The results at low and high workloads also show that the approach is feasible both for workflow topologies derived from real benchmark applications and for complex randomly generated topologies composing a high number of services, positively answering RQ4 and RQ5.

C. Discussion

Our method is open to deployment in a real scenario. Specifically, implementation in a virtualized or containerized environment (e.g., Docker) would require instantiating a container for each elementary service of the workflow, and exclusively provisioning it with a fixed amount of CPU resources defined by our technique. This implementation would enable us to test our approach in the fixed-res case (i.e., the approach uses an assigned total resource amount) and custom-res case (i.e., the approach uses its actual minimum resource amount needed to fit the workflow SLO), both at low and at high workloads. Conversely, testing our approach in the custom-res-hs scenario

(i.e., with a horizontal scaler managing replicas of elementary services) would require implementation of our approach in a container orchestration system (e.g., Kubernetes). In this case, implementation would require supplying each container instantiated by the horizontal scaler for the same elementary service with resources defined by our approach for that service.

In heterogeneous cloud environments [72], [73], [74], the characteristics of the infrastructure and the workload of workflow requests may vary over time. For requests of a *single* workflow addressed in this paper, changes in the workload of requests would not require to re-execute our approach, as the derivation of provisioning is independent of the workload. Conversely, changes in the characteristics of CPU resources would require to profile again the durations of elementary services for any provisioning (i.e., Monitor phase in Fig. 1) and to re-compute the provisioning for each service (i.e., Knowledge, Analyze, Plan, and Execute phases in Fig. 1). We could manage requests of *multiple* workflows by considering the request rate to each service in the provisioning algorithm, requiring us to re-run our approach in the case of changes in the workload of requests (in addition to the case of changes in the CPU characteristics).

For large-scale cloud systems, various works report that CPU utilization is non-linear in provisioned resources [75], [76], mainly due to the high variability of workload and the influence of queued requests, making it difficult to estimate how response times of services scale with computing resources. Therefore, the high-workload scenarios considered in the experiments are especially challenging for our approach, given that they consider queue effects and different relations between service response times and provisioned resources. Overall, the achieved results are interesting and promising for future works.

VIII. CONCLUSION

We presented a coordinated approach to provisioning of elementary services with generally distributed durations, composed in workflows with the topology of a DAG, subject to an SLO on the E2E response time distribution. The approach is open to many extensions. Specifically, we could modify our performance model to represent piecewise linear or inhomogeneous linear relations, thus modeling non-parallelizable fractions of the job size, service start-up times, and communication costs. We could use a different semantics for split-merge constructs, modeling resources accrued on-demand and by reservation, and other relations of stochastic ordering between the SLO and the workflow E2E response time. We could manage integer resource amounts to implement horizontal scaling and, to cope with limited domain knowledge, we could infer the workflow topology by observing durations of (unknown) service compositions for different provisioning. We could extend the approach also to consider requests of multiple workflows with shared services, exploiting the rate of requests to each service to determine the provisioning. Future work notably includes implementation and testing of our approach within a container orchestration system, considering workflows of benchmark microservices applications [65], [77].

REFERENCES

- [1] E. Van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A SPEC RG cloud group's vision on the performance challenges of FaaS cloud architectures," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, 2018, pp. 21–24.
- [2] K. Jiang, H. Zhou, X. Chen, and H. Zhang, "Mobile edge computing for ultra-reliable and low-latency communications," *IEEE Commun. Standards Mag.*, vol. 5, no. 2, pp. 68–75, Jun. 2021.
- [3] H. T. Malazi et al., "Dynamic service placement in multi-access edge computing: A systematic literature review," *IEEE Access*, vol. 10, pp. 32639–32688, 2022.
- [4] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Proc. 3rd Workshop Workflows Support Large-Scale Sci.*, 2008, pp. 1–10.
- [5] Z. Ahmad et al., "Scientific workflows management and scheduling in cloud computing: Taxonomy, prospects, and challenges," *IEEE Access*, vol. 9, pp. 53491–53508, 2021.
- [6] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 805–825.
- [7] H. Qiu et al., "Is function-as-a-service a good fit for latency-critical services?," in *Proc. 7th Int. Workshop Serverless Comput.*, 2021, pp. 1–8.
- [8] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time operating systems," in *Proc. Real-Time Syst. Symp.*, 1985, pp. 112–122.
- [9] J. Rahman and P. Lama, "Predicting the end-to-end tail latency of containerized microservices in the cloud," in *Proc. Int. Conf. Cloud Eng.*, 2019, pp. 200–210.
- [10] E. B. Lakew, C. Klein, F. Hernandez-Rodriguez, and E. Elmroth, "Towards faster response time models for vertical elasticity," in *Proc. IEEE/ACM Int. Conf. Utility Cloud Comput.*, 2014, pp. 560–565.
- [11] S. Farokhi, E. B. Lakew, C. Klein, I. Brandic, and E. Elmroth, "Coordinating CPU and memory elasticity controllers to meet service response time constraints," in *Proc. Int. Conf. Cloud Autonomic Comput.*, 2015, pp. 69–80.
- [12] K. Salah, K. Elbadawi, and R. Boutaba, "An analytical model for estimating cloud resources of elastic services," *J. Netw. Sys. Manag.*, vol. 24, pp. 285–308, 2016.
- [13] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola, "MOSES: A framework for QoS driven runtime adaptation of service-oriented systems," *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1138–1159, Sep./Oct. 2012.
- [14] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–33, 2018.
- [15] J. Dogani, R. Namvar, and F. Khunjush, "Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey," *Comput. Commun.*, vol. 209, pp. 120–150, 2023.
- [16] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Proc. IEEE Netw. Operations Manage. Symp.*, 2012, pp. 204–212.
- [17] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A hybrid, proactive auto-scaling mechanism on a level-playing field," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 800–813, 2018.
- [18] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads," in *Proc. 7th Int. Conf. Autonomic Comput.*, 2010, pp. 21–30.
- [19] S. J. Malkowski, M. Hedwig, J. Li, C. Pu, and D. Neumann, "Automated control for elastic n-tier workloads based on empirical modeling," in *Proc. ACM Int. Conf. Autonomic Comput.*, 2011, pp. 131–140.
- [20] I. Gergin, B. Simmons, and M. Litoiu, "A decentralized autonomic architecture for performance control in the cloud," in *Proc. Int. Conf. Cloud Eng.*, 2014, pp. 574–579.
- [21] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janeczek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Gener. Comput. Syst.*, vol. 27, no. 6, pp. 871–879, 2011.
- [22] U. Sharma, P. Shenoy, and D. F. Towsley, "Provisioning multi-tier cloud applications using statistical bounds on sojourn time," in *Proc. Int. Conf. Autonomic Comput.*, 2012, pp. 43–52.
- [23] R. Han, M. M. Ghanem, L. Guo, Y. Guo, and M. Osmond, "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications," *Future Gener. Comput. Syst.*, vol. 32, pp. 82–98, 2014.
- [24] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "Generating adaptation policies for multi-tier applications in consolidated server environments," in *Proc. Int. Conf. Autonomic Comput.*, 2008, pp. 23–32.
- [25] P. D. Kaur and I. Chana, "A resource elasticity framework for QoS-aware execution of cloud applications," *Future Gener. Comput. Syst.*, vol. 37, pp. 14–25, 2014.
- [26] P. Lama and X. Zhou, "Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee," in *Proc. IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2010, pp. 151–160.
- [27] J. Liu, S. Zhang, Q. Wang, and J. Wei, "Coordinating fast concurrency adapting with autoscaling for SLO-Oriented web applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3349–3362, Dec. 2022.
- [28] E. Incerto, M. Tribastone, and C. Trubiani, "Combined vertical and horizontal autoscaling through model predictive control," in *Proc. Int. Conf. Parallel Distrib. Comput.*, 2018, pp. 147–159.
- [29] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Proc. 12th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, pp. 644–651, 2012.
- [30] Q. Wang, H. Chen, S. Zhang, L. Hu, and B. Palanisamy, "Integrating concurrency control in N-tier application scaling management in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 855–869, Apr. 2019.
- [31] L. Yazdanov and C. Fetzer, "Vertical scaling for prioritized VMs provisioning," in *Proc. Int. Conf. Cloud Green Comput.*, 2012, pp. 118–125.
- [32] F. Filippini et al., "Figaro: Reinforcement learning management across the computing continuum," in *Proc. 3rd Workshop Distrib. Mach. Learn. Intell. Comput. Continuum*, 2023, pp. 1–8.
- [33] E. Incerto, R. Pizzoli, and M. Tribastone, "μOpt: An efficient optimal autoscaler for microservice applications," in *Proc. Int. Conf. Autonomic Comput. Self-Organizing Syst.*, 2023, pp. 67–76.
- [34] A. U. Gias, G. Casale, and M. Woodside, "ATOM: Model-driven autoscaling for microservices," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1994–2004.
- [35] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: A survey," *J. Supercomputing*, vol. 71, pp. 3373–3418, 2015.
- [36] M. Masdari, S. ValiKardan, Z. Shahi, and S. I. Azar, "Towards workflow scheduling in cloud computing: A comprehensive analysis," *J. Netw. Comput. Appl.*, vol. 66, pp. 64–82, 2016.
- [37] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, 1999.
- [38] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing," in *Metaheuristics for Scheduling in Distributed Computing Environments* (Studies in Computational Intelligence, SCI). 2008, vol. 146, pp. 173–214.
- [39] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1344–1357, May 2016.
- [40] F. Guo, L. Yu, S. Tian, and J. Yu, "A workflow task scheduling algorithm based on the resources' fuzzy clustering in cloud computing environment," *Int. J. Commun. Syst.*, vol. 28, no. 6, pp. 1053–1067, 2015.
- [41] R. Singh and S. Singh, "Score based deadline constrained workflow scheduling algorithm for cloud systems," *Int. J. Cloud Computing: Serv. Archit.*, vol. 3, no. 6, pp. 31–41, 2013.
- [42] F. Xhafa and A. Abraham, *Metaheuristics for Scheduling in Distributed Computing Environments*. Berlin, Germany: Springer, 2008.
- [43] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. Yang, "A compromised-time-cost scheduling algorithm in swindow-C for instance-intensive cost-constrained workflows on a cloud computing platform," *Int. J. High Perf. Comput. Appl.*, vol. 24, no. 4, pp. 445–456, 2010.
- [44] L. F. Bittencourt and E. R. M. Madeira, "HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds," *J. Internet Serv. Appl.*, vol. 2, pp. 207–227, 2011.
- [45] S. Kardani-Moghaddam, R. Buyya, and K. Ramamohanarao, "ADRL: A hybrid anomaly-aware deep reinforcement learning-based resource scaling in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 514–526, Mar. 2021.
- [46] G. Garbi, E. Incerto, and M. Tribastone, "μP: A development framework for predicting performance of microservices by design," in *Proc. Int. Conf. Cloud Comput.*, 2023, pp. 178–188.
- [47] G. Liu, H. Shen, and H. Wang, "An economical and SLO-guaranteed cloud storage service across multiple cloud service providers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2440–2453, Sep. 2017.

- [48] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. ACM Symp. Cloud Comput.*, 2011, pp. 1–14.
- [49] L. Carnevali, M. Paolieri, B. Picano, R. Reali, L. Scommegna, and E. Vicario, "A quantitative approach to coordinated scaling of resources in complex cloud computing workflows," in *Proc. Eur. Workshop Perform. Eng.*, 2023, pp. 309–324.
- [50] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.
- [51] W. M. van Der Aalst, A. H. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, pp. 5–51, 2003.
- [52] R. A. Sahner and K. S. Trivedi, "Performance and reliability analysis using directed acyclic graphs," *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 10, pp. 1105–1114, 1987.
- [53] K. S. Trivedi and R. Sahner, "SHARPE at the age of twenty two," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, pp. 52–57, Mar. 2009.
- [54] L. Carnevali, R. Reali, and E. Vicario, "Eulero: A tool for quantitative modeling and evaluation of complex workflows," in *Proc. Int. Conf. Quantitative Eval. Syst.*, 2022, pp. 255–272.
- [55] M. Dumas and A. H. Ter Hofstede, "UML activity diagrams as a workflow specification language," in *Proc. Int. Conf. Unified Model. Lang.*, 2001, pp. 76–90.
- [56] H. Eshuis, "Semantics and verification of UML activity diagrams for workflow modelling," Ph.D. dissertation, Univ. Twente, 2002.
- [57] B. Berg, J. L. Dorsman, and M. Harchol-Balter, "Towards optimality in parallel scheduling," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 2, pp. 40:1–40:30, 2017.
- [58] Z. Li, B. Berg, A. Mukhopadhyay, and M. Harchol-Balter, "How to rent GPUs on a Budget," 2024, [arXiv:2406.15560](https://arxiv.org/abs/2406.15560).
- [59] L. Carnevali, R. Reali, and E. Vicario, "Compositional evaluation of stochastic workflows for response time analysis of composite web services," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, 2021, pp. 177–188.
- [60] Z. Zheng and M. R. Lyu, "WS-DREAM: A distributed reliability assessment mechanism for web services," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, 2008, pp. 392–397.
- [61] L. Wang, E. Egorova, and A. Mokryakov, "Development of hypergraph theory," *J. Comput. Syst. Sci. Int.*, vol. 57, pp. 109–114, 2018.
- [62] L. Carnevali, M. Paolieri, R. Reali, and E. Vicario, "Compositional safe approximation of response time probability density function of complex workflows," *ACM Trans. Model. Comput. Simul.*, vol. 33, 2023, Art. no. 16.
- [63] A. Mirhosseini, S. Elnikety, and T. F. Wenisch, "Parslo: A gradient descent-based approach for near-optimal partial SLO allotment in microservices," in *Proc. ACM Symp. Cloud Comput.*, 2021, pp. 442–457.
- [64] V. M. Bhasi, J. R. Gunasekaran, P. Thinakaran, C. S. Mishra, M. T. Kandemir, and C. Das, "Kraken: Adaptive container provisioning for deploying dynamic dags in serverless platforms," in *Proc. ACM Symp. Cloud Comput.*, 2021, pp. 153–167.
- [65] Y. Gan et al., "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proc. Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, pp. 3–18.
- [66] Pegasus, "Pegasus workflow repository," 2025. [Online]. Available: https://pegasus.isi.edu/workflow_gallery
- [67] E. Deelman et al., "The evolution of the Pegasus workflow management software," *Comput. Sci. Eng.*, vol. 21, no. 4, pp. 22–36, 2019.
- [68] J. W. Tukey et al., *Exploratory Data Analysis*, vol. 2. Reading, MA, 1977.
- [69] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *J. Amer. Stat. Assoc.*, vol. 47, no. 260, pp. 583–621, 1952.
- [70] R. E. Tarone, "Testing the goodness of fit of the binomial distribution," *Biometrika*, vol. 66, no. 3, pp. 585–590, 1979.
- [71] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. Spring Joint Comput. Conf.*, 1967, pp. 483–485.
- [72] H. Qiu et al., "Flash: Fast model adaptation in ML-centric cloud platforms," *Mach. Learn. Syst.*, vol. 6, pp. 524–544, 2024.
- [73] K. Hazelwood et al., "Applied Machine Learning At Facebook: A Datacenter Infrastructure Perspective," in *Proc. Int. Symp. High Perform. Comput. Architecture*, 2018, pp. 620–629.
- [74] S. Kanev et al., "Profiling a warehouse-scale computer," in *Proc. Int. Symp. Comput. Archit.*, 2015, pp. 158–169.
- [75] Z. Wang et al., "Deepscaling: Microservices autoscaling for stable CPU utilization in large scale cloud systems," in *Proc. 13th Symp. Cloud Comput.*, 2022, pp. 16–30.
- [76] H. Qiu et al., "Simppo: A scalable and incremental online learning framework for serverless resource management," in *Proc. Symp. Cloud Comput.*, 2022, pp. 306–322.
- [77] X. Zhou et al., "Benchmarking microservice systems for software engineering research," in *Proc. Int. Conf. Softw. Eng.: Companion Proc.*, 2018, pp. 323–324.



Laura Carnevali (Member, IEEE) received the MS degree in computer engineering and the PhD degree in informatics, multimedia, and telecommunications engineering from the University of Florence, Florence, Italy, in 2006 and 2010, respectively. She is an associate professor of computer science with the School of Engineering, University of Florence, Italy. Her research is focused on correctness verification and performance evaluation of real-time systems, with focus on stochastic characterization of timed models.



Marco Paolieri (Member, IEEE) received the the MS degree in computer engineering and PhD degree in computer science, systems, and telecommunications from the University of Florence, Italy, in 2011 and 2015, respectively. He is a senior research associate with the University of Southern California, Los Angeles, USA. His research interests include focus on stochastic modeling and quantitative evaluation of performance and reliability in concurrent and distributed systems.



Riccardo Reali received the bachelor and master degrees in information engineering and the PhD degree in smart computing from the University of Florence. He is a postdoctoral researcher with the University of Florence. His scientific activity is focused on quantitative analysis of non-Markovian through compositional and heuristics methods, software engineering methodologies, and software architectures.



Leonardo Scommegna (Member, IEEE) received the PhD degree in smart computing from the University of Florence, Italy, in 2023. He is assistant professor of computer science with the Department of Information Engineering, University of Florence, Italy. His research interests include area of software engineering, with particular emphasis on software architectures, software testing, and model-based development.



Enrico Vicario (Member, IEEE) is a professor of computer science and engineering and head of the Department of Information Engineering, University of Florence, Italy. His research interests include the area of software engineering, with a present focus on quantitative evaluation of stochastic models, software architectures and methodologies, and on their connection through model driven engineering practices.