Compositional Coordinated Resource Provisioning for Workflows with Stochastic Durations in Kubernetes

Tommaso Botarelli
tommaso.botarelli@unifi.it
University of Florence
Florence, Italy

Laura Carnevali
laura.carnevali@unifi.it
University of Florence
Florence, Italy

Leonardo Scommegna
leonardo.scommegna@unifi.it
University of Florence
Florence, Italy

Enrico Vicario
enrico.vicario@unifi.it
University of Florence
Florence, Italy

Abstract

In workflows with stochastic durations, the end-to-end (E2E) response time distribution is jointly determined by several factors, including the workflow topology, the response time distribution of elementary services, and their sensitivity to resource scaling. This complexity is further exacerbated when workflows are deployed on microservices architectures, where additional factors related to the orchestration infrastructure may affect the E2E response time.

In this paper, we apply a state-of-the-art resource provisioning method to a real container orchestration system. Specifically, the method coordinates resource allocation for elementary services by jointly considering the factors mentioned above. We consider workflows with randomly generated topology and with elementary service durations drawn from a data set used in the literature. We implement these workflows as microservices and we deploy them on Kubernetes using the proposed provisioning strategy. Experimental results demonstrate the effectiveness of the approach compared to alternative baselines, under both low and high workload conditions.

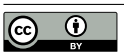
CCS Concepts

• **Computer systems organization → Cloud computing**; • **Software and its engineering → Software performance**; **Software system models**.

Keywords

Coordinated resource provisioning, microservices workflows, end-to-end response time distribution, non-Markovian durations.

ACM Reference Format:
Tommaso Botarelli, Laura Carnevali, Leonardo Scommegna, and Enrico Vicario. 2026. Compositional Coordinated Resource Provisioning for Workflows with Stochastic Durations in Kubernetes. In *Companion of the 17th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '26), May 04–08, 2026, Florence, Italy.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3777911.3800627



ICPE Companion '26, Florence, Italy

ACM ISBN 979-8-4007-2326-1/2026/05
https://doi.org/10.1145/3777911.3800627

1 Introduction

In the landscape of modern software systems, the paradigm shift toward cloud-native architectures has fundamentally altered how applications are designed, deployed, and managed. Monolithic applications have largely given way to microservices architectures, where complex functionalities are delivered through the orchestration of loosely coupled, atomic, and heterogeneous services [13, 15]. While offering significant agility and scalability, this decomposition introduces substantial complexity in resource management. A single user request often triggers a workflow of multiple interacting services [1, 4, 9]. Thus, the End-to-End (E2E) response time experienced by the user is not merely the sum of deterministic execution times, rather it is a random variable jointly determined by the workflow topology, the response time distribution of elementary services, and their individual sensitivity to resource scaling.

Resource provisioning in this context is a critical challenge. For providers and consumers operating on a pay-per-use basis, the goal is to optimize resource allocation to maximize performance without over-provisioning [12]. Remarkable examples in the literature address performance modeling with validation against real implementations, for instance, by exploiting the Layered Queueing Network (LQN) paradigm [6, 11]. However, these approaches often rely on the assumption of ample resource availability or depend on periodic re-modulation of provisioning to adjust to runtime variability [10]. Consequently, defining the optimal configuration under strict resource scarcity presents distinct challenges. The stochastic nature of microservices causes fluctuation in the execution times, and the dependencies within the workflow topology can amplify delays. Furthermore, theoretical models often assume idealized environments. When the workflows are deployed on real-world orchestration platforms like Kubernetes, additional factors related to the infrastructure, such as networking overhead, container startup latency, and resource scheduling and sharing, can significantly alter the expected E2E performance. There is, therefore, a pressing need to validate theoretical provisioning strategies against the reality of actual container orchestration systems.

In this paper, we bridge the gap between theoretical resource modeling and practical deployment, by applying a coordinated resource provisioning method for stochastic workflows, which we recently introduced [3], to a real Kubernetes environment. In particular, we investigate how well the coordinated allocation strategy, which jointly accounts for topology, service duration distributions,

and scaling sensitivity, performs when subjected to the operational constraints of a real production environment.

We implement a diverse set of workflows as actual microservices, generating topologies randomly to ensure statistical robustness and drawing elementary service durations from established datasets in the literature [16]. We then deploy these workflows on Kubernetes, using our provisioning strategy to dictate the resource limits of each container. By conducting experiments under both low and high workload conditions, we show that our theoretically grounded approach translates into tangible performance gains in a real system. The experimental results indicate that our strategy effectively manages the trade-off between resource usage and response time, consistently outperforming alternative ablation strategies. This outcome validates that considering the stochastic durations and topological dependencies of workflows is essential for efficient resource provisioning in cloud-native environments.

2 Background

In this section, we describe the considered class of workflows (Section 2.1) and the resource provisioning technique of [3] (Section 2.2).

2.1 Workflows with stochastic durations

Fig. 1 shows an example of workflow topology, specified using the notation of the activity diagram of UML [5]: *elementary actions* represent elementary activities (i.e., `A11`, `A12`, `A21`, and `A22`), each consisting of a CPU bound task, whose completion time depends on the provisioned CPU amount; *structured actions* compose other actions, which can be elementary actions or structured actions themselves. We consider three composition patterns: *i*) the *sequence* pattern, modeling the sequential execution of actions (i.e., `A1`); *ii*) the *fork-join* pattern, modeling the concurrent execution of actions (i.e., `A`), and *iii*) the *split-merge* pattern, modeling the exclusive execution of actions, each with an assigned probability (i.e., `A2`).

The workflow E2E response time is a random variable determined by the durations of the elementary actions and the topology that composes them. The E2E response time of an elementary action a provisioned with CPU amount r is a random variable $\mathbb{T}(a, r)$:

$$\mathbb{T}(a, r) = \frac{\mathbb{J}(a)}{r} \tag{1}$$

where $\mathbb{J}(a)$ is the *job size* of a, i.e., the duration of the CPU bound task when a is provisioned with a unit of CPU resource [2, 3, 8].

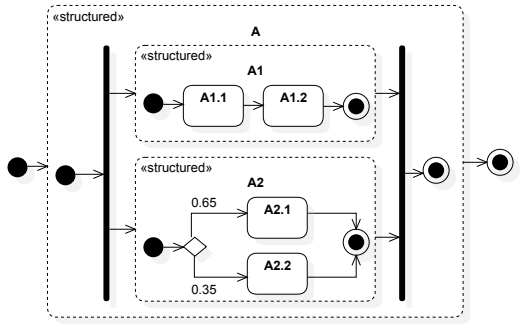


Figure 1: UML activity diagram of a workflow.

The E2E response time of a structured action a is a random variable depending on the composition pattern. Let a be provisioned with CPU amount r and compose actions $a_1, \dots, a_N$ provisioned with CPU amount $r_1, \dots, r_N$, respectively, such that $r = \sum_{n=1}^{N} r_n$:

- If a is the sequence of $a_1, \dots, a_N$, $\mathbb{T}(a, r) = \sum_{n=1}^{N} \mathbb{T}(a_n, r_n)$ is the *sum* of the E2E response times of $a_1, \dots, a_N$.
- If a is the fork-join of $a_1, \dots, a_N$, $\mathbb{T}(a, r) = \max_{n=1}^{N} \mathbb{T}(a_n, r_n)$ is the *maximum* of the E2E response times of $a_1, \dots, a_N$.
- If a is the split-merge of $a_1, \dots, a_N$ with probabilities $p_1, \dots, p_N$, respectively, such that $\sum_{n=1}^{N} p_n = 1$, then $\mathbb{T}(a, r) = \sum_{n=1}^{N} p_n \cdot \mathbb{T}(a_n, r_n)$ is the *mixture* of the E2E response times of $a_1, \dots, a_N$ weighted by probabilities $p_1, \dots, p_N$, respectively.

2.2 Compositional coordinated provisioning

Given a workflow and a total resource capacity R, our approach derives a resource allocation with the aim of minimizing the workflow E2E response time. The approach models the workflow as a *structure tree*, i.e., a tuple $T = \langle V, E, v_0 \rangle$ where N is the set of nodes, E is the set of edges, and $v_0 \in V$ is the root node. Nodes are partitioned into leaf nodes, modeling elementary actions (in the example: `A1.1`, `A1.2`, `A2.1`, `A2.2`), and internal nodes, modeling structured actions. Edges represent composition relations between actions. We derive the structure tree of a workflow in a straightforward manner, by visiting the UML activity diagram starting from the action modeling the whole workflow, and by translating elementary actions into leaf nodes, and structured actions into internal nodes.

Our strategy performs a top-down traversal of the tree, starting from the root node, which is assigned a CPU budget equal to the total amount R. Each intermediate node is allocated a CPU quantity equal to the amount required for its subtree. For each node v corresponding to an action a provisioned with CPU amount R_v, the CPU allocation to its child nodes is based on a proxy metric of the response time $\gamma(\mathbb{T}(a, r))$:

- If v is a sequence of nodes $v_1, \dots, v_N$, the CPU amount R_v is partitioned so as to minimize $\sum_{n=1}^{N} \gamma(\mathbb{T}(a_n, r_n))$. By employing the Lagrangian formulation subject to the constraint $\sum_{n=1}^{N} r_n = R_v$, the optimal resource allocation r'_n is:

$$r'_n = \frac{\sqrt{\gamma(\mathbb{T}(a_n, r_n))}}{\sum_{i=1}^{N} \sqrt{\gamma(\mathbb{T}(a_n, r_i))}} \cdot R_v \qquad \forall\, n \in \{1, \dots, N\} \tag{2}$$

- If v is a split-merge of nodes $v_1, \dots, v_N$ with associated probabilities $p_1, \dots, p_n$, the CPU amount R_v is partitioned so as to minimize $\sum_{n=1}^{N} p_n \cdot \gamma(\mathbb{T}(a_n, r_n))$. By employing the Lagrangian formulation subject to the constraint $\sum_{n=1}^{N} r_n = R_v$, the optimal resource allocation r'_n is:

$$r'_n = \frac{\sqrt{\gamma(\mathbb{T}(a_n, r_n))}}{\sum_{i=1}^{N} \sqrt{\gamma(\mathbb{T}(a_n, r_i))}} \cdot R_v \qquad \forall\, n \in \{1, \dots, N\} \tag{3}$$

- If v is a fork-join of nodes $v_1, \dots, v_N$, we replace it into an equivalent binary composition of fork-join nodes, i.e., node v becomes the fork-join of nodes v_1 and v'_2, where v'_2 is the fork-join of nodes v_2 and v'_3, and so on, until we define node v'_{N-1} as the fork-join of nodes v_{N-1} and v_N. So the resources of the two child nodes r'_1 and r'_2 are allocated so as to balance

the proxy metric of the E2E response time:

$$r'_n = \frac{\gamma(\mathbb{T}(a_n, r_n))}{\gamma(\mathbb{T}(a_n, r_n)) + \gamma(\mathbb{T}(a_{\bar{n}}, r_{\bar{n}}))} \cdot R_v \qquad n \in \{1, 2\} \tag{4}$$

where $\bar{n}$ indicates the opposite child node.

In our experimental evaluation, the proxy metric γ is the expected value $\mathbb{E}$. However, the algorithm is equally applicable with any other metric (e.g., percentiles, standard deviation, or a mixture of metrics) to target different optimization objectives.

3 Experimental validation

This section evaluates the *performance efficiency*, as defined by the ISO/IEC 25010 standard [7], of our CPU provisioning method. We test the approach in a real Kubernetes-based environment to answer two main research questions: does our method achieve a better response time distribution than the baselines in (**RQ1**) low-workload scenarios and (**RQ2**) high-workload scenarios?

In the following, we describe the experimental setup (Section 3.1), define the alternative baselines (Section 3.2), and introduce the quantitative measure used for comparison (Section 3.3).

3.1 Experimental setup

The experiments were performed on randomly generated microservice workflows, with topologies of varying complexity. The workflows feature a tree structure, which represents the most prevalent pattern in microservice architectures [9]. The generation process follows a top-down approach, parameterized by a target tree height D and a maximum number of children per node $B \geq 2$. At each step, a composition pattern is randomly selected along with a random number of children $n \in \{2, ..., B\}$. The process continues until the current height d reaches D. At this point, elementary actions are generated, and the tree construction terminates. Job sizes for the elementary actions were sampled from 100 histograms of microservice response times in the WS-Dream dataset [16], processed via the inter-quartile range rule [14] and scaled to be within $[10, 100]$ ms. The obtained workflows consist of 5 to 20 elementary actions.

The experiments were performed on 8 workflows generated using $D \in \{2, 3, 4\}$ and $B \in \{3, 4\}$ (see Table 1), provisioned by three baselines (Section 3.2) and by our approach. The provisioning derivation for our approach took nearly 100 ms. Then, the *YAML* files required for deployment in a Kubernetes cluster were automatically generated from the resulting topologies. The Kubernetes services use a custom Docker image that performs a "busy sleep" operation, consuming CPU resources according to the associated job size distribution $\mathbb{J}$. After each busy sleep operation, one or more services are invoked based on the topology. Once the services are deployed, a Python script handles load generation. and the E2E response times are collected. The experiments were performed on a single node equipped with Intel Xeon Gold 6238R processor with 2.2 GHz frequency and 135 GB RAM, using Minikube as Kubernetes environment.

3.2 Baseline resource provisioning techniques

To evaluate the proposed approach, we consider three alternative baseline provisioning strategies defined in [3],:

- **Completely Agnostic Provisioning (CAP)**. Given a workflow W with N elementary actions and a set of resources R, this method allocates R equally among all elementary tasks: $\forall n \in \{1, \ldots, N\}\ r_n = R/N$.
- **Topology Driven Provisioning (TDP)**. Given a workflow W with N elementary actions and a set of resources R, this method generates a workflow W' derived from W by replacing the job sizes of each elementary action a_n with a unitary job size $\mathbb{J}(a_n) = Det(1)\ \forall n \in \{1, \ldots, N\}$. Then, the allocation follows the proposed methodology (Section 2.2).
- **Balanced Duration Driven Provisioning (BDDP)**. Given a workflow W with N elementary actions and a set of resources R, this method constructs a workflow W' as a single fork-join node of all the elementary actions of W. It allocates resources to the elementary actions proportionally by considering a proxy of the job size γ: $r_n = \frac{\gamma(\mathbb{T}(a_n, r_n))}{\sum_{i=1}^{N} \gamma(\mathbb{T}(a_i, r_i))} \cdot R$. In our experimentation, γ is the expected value $\mathbb{E}$.

These baselines allow us to evaluate our approach against methodologies that disregard the topology of the workflow (BDDP), the response times of the elementary actions (TDP), or both (CAP).

3.3 Quantitative measure of interest

We assess the relative performance of our approach and each baseline techniques in terms of workflow response time distribution by using the *pairwise comparison dominance*. Let $\tilde{\tau}$ and τ two random variables the pairwise comparison dominance $D(\tilde{\tau}, \tau)$ between $\tilde{\tau}$ and τ is an average measure of how many times $\tilde{\tau}$ anticipates τ:

$$D(\tilde{\tau}, \tau) := \text{Prob}\{\tilde{\tau} \leq \tau\} = \int_0^\infty (1 - F_\tau(t)) \cdot f_{\tilde{\tau}}(t) dt. \tag{5}$$

where $f_\tau(t)$ and $F_\tau(t)$ represent respectively the PDF and the CDF of the random variable τ. Starting from the pairwise comparison dominance we compute the *pairwise comparison dominance deviation* (dominance deviation for short):

$$\Delta(\tilde{\tau}, \tau) = D(\tilde{\tau}, \tau) - \frac{1}{2} \tag{6}$$

If $\Delta(\tilde{\tau}, \tau) \in (0, \frac{1}{2}]$ then $\tilde{\tau}$ anticipates τ more than viceversa and if $\Delta(\tilde{\tau}, \tau) \in [-\frac{1}{2}, 0)$ then τ anticipates $\tilde{\tau}$ more than viceversa. In the practice from the collected E2E workflow response time, we construct the Empirical Cumulative Distribution Functions (ECDFs) $\widehat{F}(t) = \frac{\#\text{samples} \leq t}{N}$ where N is the total number of samples. Therefore, starting from the collected data, the Empirical Probability Density Functions (EPDFs) $\widehat{f}(t)$ can be obtained by discretizing the time range into bins and computing the relative frequency of samples within each interval. Thus, we calculate $D(\tilde{\tau}, \tau)$ and $\Delta(\tilde{\tau}, \tau)$ by replacing the PDF and CDF in Eq. (5) with their respective empirical formulations and performing numerical integration.

4 Experimental results

Low-workload scenario. We perform the low-workload experiments by sending a workflow request only once the preceding request has been fully served. Table 1 shows the dominance deviation achieved by our approach with respect to each baseline: for each workflow, our approach provides the best performance if it

Table 1: For each workflow model and workload conditions, dominance deviation achieved by our approach with respect to each baseline. Our approach outperforms in case of positive dominance deviation value.

workflow	structure	low-workload conditions			high-workload conditions		
		Δ(ours, CAP)	Δ(ours, TDP)	Δ(ours, BDDP)	Δ(ours, CAP)	Δ(ours, TDP)	Δ(ours, BDDP)
M1	D2-B3	0.0374	0.1796	0.1194	0.0363	0.2739	0.0923
M2	D2-B3	0.0097	0.1569	0.2086	0.0414	0.2556	0.3740
M3	D2-B4	0.0335	0.3073	0.1115	0.0210	0.4954	0.1416
M4	D2-B4	0.0038	0.1439	0.1754	0.0632	0.4920	0.3533
M5	D3-B3	0.0354	0.4136	0.2613	0.0164	0.4518	0.2791
M6	D3-B3	0.0668	0.0241	0.2205	0.0850	-0.0257	0.2074
M7	D3-B4	0.0712	0.1809	0.2124	0.0927	0.2183	0.2382
M8	D4-B3	0.0827	0.1637	0.1714	0.0853	0.1809	0.2054
Δ Mean		0.0426	0.1963	0.1850	0.0551	0.2928	0.2364

achieves positive dominance deviation with respect to each baseline, while the baselines provide better performance with respect to each other in decreasing order of dominance deviation (i.e., lower dominance deviation indicates better performance).

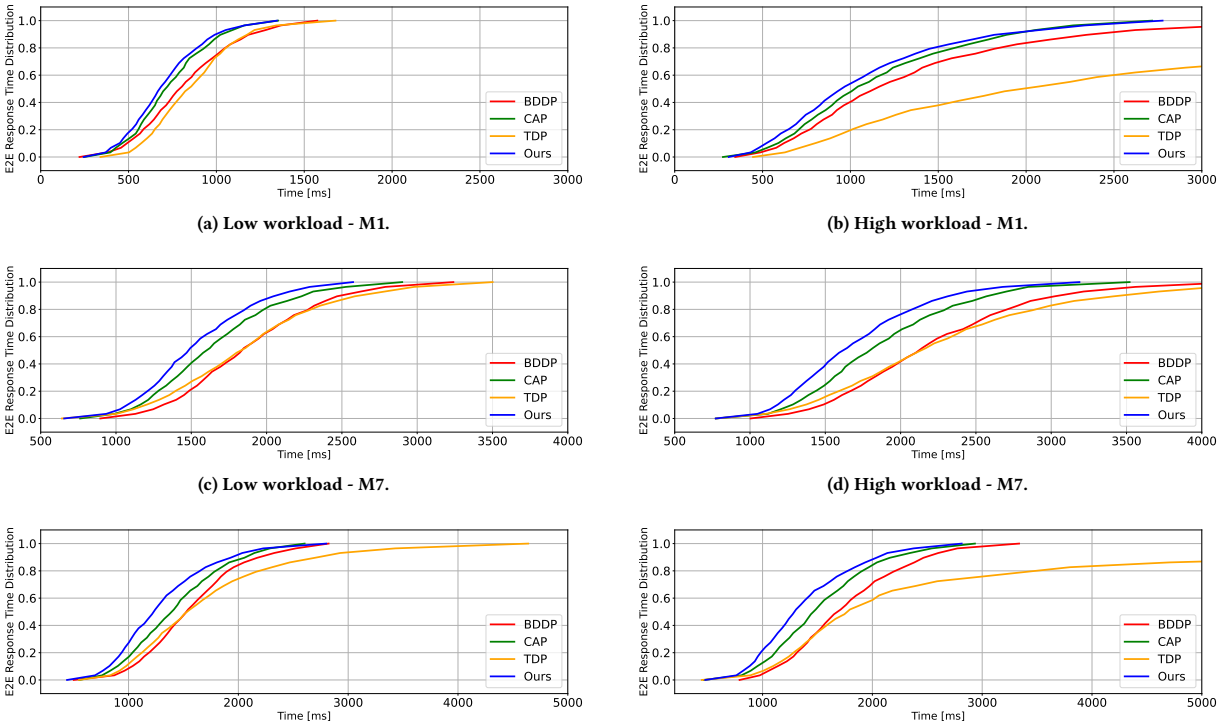
Under low-workload scenarios, the proposed approach demonstrates robust and consistent superiority over all competing methodologies. Our approach achieves the most significant performance gains against the TDP baseline, with Δ(ours, TDP) reaching as high as 0.4136 (M5). This indicates that strategies based exclusively on topology awareness are insufficient to achieve the performance achievements of the proposed method. The proposed methodology consistently outperforms BDDP, achieving deviation scores between 0.11 and 0.26. This highlights the limitations of provisioning strategies that depend exclusively on duration analysis. Conversely to the topology-only (TDP) baseline, the BDDP results indicate that duration data lack the necessary context when used in isolation. This outlines how combining the service duration knowledge with the workflow topology is indispensable for achieving superior provisioning results in terms of E2E distribution. Surprisingly the margins against the CAP baseline are generally tighter with a mean dominance deviation of 0.0426. This results from the choice of the distributions for the elementary actions. Indeed it is critical to contextualize these results within the mechanics of Kubernetes resource scheduling. CPU limits are enforced via the Completely Fair Scheduler (CFS) bandwidth control, which typically operates on a 100 ms period. Consequently, a limit of 500 mCPU grants a container 50 ms of runtime per period. For microservices characterized by short processing times (e.g. $< 100\,ms$), the execution can complete within the initial time slice. In such "short-job" scenarios, processing time becomes invariant to provisioning levels, allocating 100 mCPU versus 1000 mCPU yields identical processing times as long as the quota is not exhausted. Consequently, although CAP demonstrates surprisingly good performance in these experiments, it might not ensure comparable stability in scenarios where processing times exhibit greater skew or longer absolute durations than those analyzed here.

Collectively these findings suggest that our approach consistently outperforms all baselines by maintaining a better response time distribution under low-workload conditions.

High-workload scenario. We perform the high-workload experiments by sending workflow requests with exponential inter-arrival time with rate $\frac{4}{5}\mu$, where μ denotes the workflow average response time measured at low workloads across the CPU allocations provided by our approach and the baselines. In this high-load scenario, our method maintains superior performance compared to the other cases. In fact, our method ranks first in terms of dominance deviation in 7 out of the 8 tested workflows. As shown in Table 1, it can be observed that the average dominance deviation for TDP and BDDP is higher under high-load conditions. This was expected, as suboptimal provisioning leads to performance degradation when the load increases, correspondingly raising the pressure on certain services that may be under-provisioned. In only one instance does our approach rank second (M6). In this specific case, the workflow features a significant presence of split-merge nodes, both at the root and within intermediate tree nodes. This implies that the topological structure plays a more critical role than task duration knowledge. Consequently, it is observed that TDP performs significantly better than BDDP in this scenario. Despite this, the proposed approach continues to outperform the other methods. As shown in Fig. 2, TDP proves to be the least stable method when transitioning from low to high workloads (M1, M7, M8). Indeed, the highly unbalanced resource provisioning causes this method to suffer significantly as the load increases, generating substantial queues in under-provisioned services and consequently degrading performance. The CAP method exhibits sustained stability, even under high workload conditions. This behavior can be attributed to the fact that the selected workload magnitude does not induce excessive stress within the system. Consequently, performance patterns observed during low-workload regimes are often mirrored in high-workload scenarios.

Except for M6, these results confirm that our approach maintains a clear performance advantage across all high-workload scenarios.

Threats to validity. Our experimental setup uses a single-node Minikube environment. This limitation is mitigated by the observation that the processing times of the individual services far exceed network communication delays. Thus, the additional latency inherent in a distributed multi-node cluster would have a negligible impact and would not affect the choices made by the provisioning



(a) Low workload - M1.

(b) High workload - M1.

(c) Low workload - M7.

(d) High workload - M7.

(e) Low workload - M8.

(f) High workload - M8.

Figure 2: Workflow E2E response time distribution achieved by our approach and the baselines.

method. Moreover, we use synthetic workflow topologies rather than those of benchmark applications. We mitigated this concern by randomly generating a substantial number of topologies, which permitted us to perform a sound experimental evaluation.

5 Conclusions

We have presented a resource provisioning strategy for microservices workflow that effectively integrates topology and processing time distribution knowledge. The experimental analysis conducted in this study suggests that in low-workload scenarios, our approach consistently achieves better performance in terms of dominance deviation compared to all baselines. The results confirm that this superiority is maintained and amplified under high-workload conditions. Collectively, these findings indicate that jointly considering the workflow topology and the duration distributions is essential for robust resource provisioning across varying load intensities.

References

[1] Kapil Bakshi. 2017. Microservices-based software architecture and approaches. In *2017 IEEE aerospace conference*. IEEE, 1–8.

[2] Benjamin Berg, Jan-Pieter L. Dorsman, and Mor Harchol-Balter. 2017. Towards Optimality in Parallel Scheduling. *Proc. ACM Meas. Anal. Comput. Syst.* 1, 2 (2017), 40:1–40:30. doi:10.1145/3154499

[3] Laura Carnevali, Marco Paolieri, Riccardo Reali, Leonardo Scommegna, and Enrico Vicario. 2025. Compositional Coordinated Resource Provisioning in Workflows With Stochastic Durations. *IEEE Trans. on Par. and Distrib. Sys.* (2025).

[4] Andrea Detti, Ludovico Funari, and Luca Petrucci. 2023. μBench: An open-source factory of benchmark microservice applications. *IEEE Transactions on Parallel and Distributed Systems* 34, 3 (2023), 968–980.

[5] Marlon Dumas and Arthur HM Ter Hofstede. 2001. UML activity diagrams as a workflow specification language. In *International conference on the unified modeling language*. Springer, 76–90.

[6] Emilio Incerto, Roberto Pizziol, and Mirco Tribastone. 2023. μOpt: An efficient optimal autoscaler for microservice applications. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 67–76.

[7] ISO/IEC. 2011. ISO/IEC 25010:2011 System and software quality models. https://www.iso.org/standard/35733.html This is a key reference for the SQuaRE standard which defines a framework for evaluating software product quality.

[8] Zhouzi Li, Benjamin Berg, Arpan Mukhopadhyay, and Mor Harchol-Balter. 2024. How to Rent GPUs on a Budget. *arXiv preprint arXiv:2406.15560* (2024).

[9] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing*. Association for Computing Machinery, New York, NY, USA, 412–426. doi:10.1145/3472883.3487003

[10] Nima Mahmoudi and Hamzeh Khazaei. 2022. Performance modeling of metric-based serverless computing platforms. *IEEE Transactions on Cloud Computing* 11, 2 (2022), 1899–1910.

[11] Giovanni Quattrocchi, Emilio Incerto, Riccardo Pinciroli, Catia Trubiani, and Luciano Baresi. 2024. Autoscaling solutions for cloud applications under dynamic workloads. *IEEE Transactions on Services Computing* 17, 3 (2024), 804–820.

[12] Sukhpal Singh and Inderveer Chana. 2016. Cloud resource provisioning: survey, status and future research directions. *Knowledge and Information Systems* 49, 3 (2016), 1005–1069.

[13] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2017. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing* 4, 5 (2017), 22–32.

[14] John W Tukey et al. 1977. *Exploratory data analysis*. Vol. 2. Reading, MA.

[15] Victor Velepucha and Pamela Flores. 2023. A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE access* 11 (2023), 88339–88358.

[16] Zibin Zheng and Michael R Lyu. 2008. Ws-dream: A distributed reliability assessment mechanism for web services. In *2008 IEEE Int.Conf. on Dependable Systems and Networks (DSN)*. IEEE.