A Trace-Based Performance Study of Autoscaling Workloads of Workflows in Datacenters

Laurens Versluis
Vrije Universiteit Amsterdam
l.f.d.versluis@vu.nl

Mihai Neacşu
Vrije Universiteit Amsterdam
m.neacsu@atlarge-research.com

Alexandru Iosup
Vrije Universiteit Amsterdam
a.iosup@vu.nl

***Abstract*—To improve customer experience, datacenter operators offer support for simplifying application and resource management. For example, running workloads of workflows on behalf of customers is desirable, but requires increasingly more sophisticated autoscaling policies, that is, policies that dynamically provision resources for the customer. Although selecting and tuning autoscaling policies is a challenging task for datacenter operators, so far relatively few studies investigate the performance of autoscaling for workloads of workflows. Complementing previous knowledge, in this work we propose the first comprehensive performance study in the field. Using trace-based simulation, we compare state-of-the-art autoscaling policies across multiple application domains, workload arrival patterns (e.g., burstiness), and system utilization levels. We further investigate the interplay between autoscaling and regular allocation policies, and the complexity cost of autoscaling. Our quantitative study focuses not only on traditional performance metrics and on state-of-the-art elasticity metrics, but also on time- and memory-related autoscaling-complexity metrics. Our main results give strong and quantitative evidence about previously unreported operational behavior, for example, that autoscaling policies perform differently across application domains and allocation and provisioning policies should be co-designed.**

I. Introduction

Many application domains of datacenter computing, from science to industrial processes to engineering, are based today on the execution of complex *workloads comprised of workflows*. To run such workloads in datacenters offering services as *clouds*, customers and providers must agree on a shared set of resource management and scheduling practices that automate the execution of many interdependent tasks, subject to Quality-of-Service (QoS) agreements. In particular, they must agree on how to continuously acquire and release resources using *autoscaling* approaches, to lower operational costs for cloud customers and to increase resource utilization for cloud operators. Although many autoscaling approaches have been proposed, currently their behavior is not studied comprehensively. This important problem contributes to the low utilization of current datacenter (and especially cloud-based) environments, as low as 6%–12% [1]–[3]. The state-of-the-art in analyzing the performance of datacenter autoscalers uses a systematic approach consisting of multiple metrics, statistically sound analysis, and various kinds of comparison tournaments [4], [5], yet lacks diversity in the application domain and insight into the interplay between components of the autoscaling system. Addressing these previously unstudied factors, in this work we propose a comprehensive study of autoscaling approaches.

We focus on workloads of workflows running in datacenters. Although the workloads of datacenters keep evolving, in our longitudinal study of datacenter workloads we have observed that many core properties persist over time [6]. Workloads exhibit often non-exponential, even bursty [7], arrival patterns, and non-exponential operational behavior (e.g., runtime distributions). Workflows are increasingly more common to use in datacenter environments, and are still rising in popularity across a variety of domains [5], [8]. Importantly, the characteristics of workflows vary widely across application domains, as indicated by the workflow size and per-task runtime reported for scientific [9], industrial processes [10], and engineering domains [11].

Managing workloads of workflows, subject to QoS requirements, is a complex activity [12]. Due to the often complex structure of workflows, resource demands can change significantly over time, dynamically as the workflow tasks are executed and reflecting the task dependencies. Thus, designing and tuning autoscaling techniques to improve resource utilization in the datacenter, while not affecting workflow performance, is non-trivial.

Studying autoscalers is a challenging activity, where innovation could appear in formulating new research questions about the laws of operation of autoscalers, or in creating adequate, reproducible experimental designs that reveal the laws. This matches challenges C6, C7, and to some extent C9 of our long-term vision on Massivizing Computer Systems [13]. In this work, we propose the following new research questions: *Does the application domain have an impact on the performance of autoscalers?*, *What is the performance of autoscalers in response to significant workload bursts?*, *What is the impact on autoscaling performance of the architecture of the scheduling system, and in particular of the (complementary) allocation policy?*, and *What is the impact on autoscaling performance of the datacenter environment?* We do not analyze further pricing strategies and models, as this is beyond of the scope of this work.

We create a simulation-based experimental environment, using trace-based, realistic workloads as input. Our choice for this setup is motivated by pragmatic and methodological reasons. Currently, no analytical model based on either state-

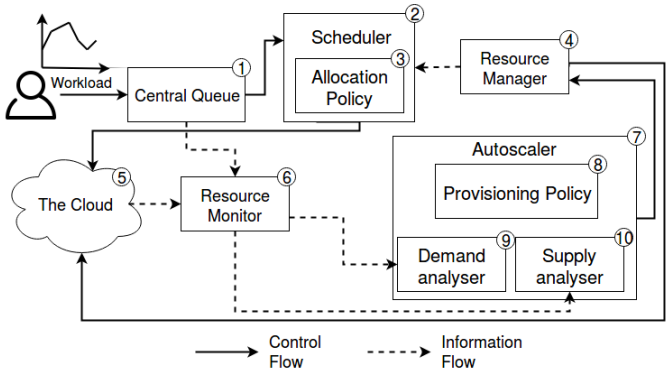

Fig. 1: The cloud-based system model, focusing on resource autoscaling, including provisioning, and allocation.

space exploration or non-state-space methods[1] has been shown able to capture the complex workload-system interplay observed in practice for the context of this work. Such analytical models are hampered, e.g., by the lack of process stationarity (i.e., burstiness and dynamic workloads) and the "curse of dimensionality" stemming from sheer scale. Similarly, conducting real-world experiments, such as our own large-scale experiments [5] is also challenging in this context, due to the high cost and long duration of experiments, and the unreproducibility of experiments or statistical inconclusiveness of reported results [4].

In summary, our contribution in this work is four-fold:

1) We are the first to investigate the impact on autoscaling performance of the application domain (Section IV). To this end, we experiment with workloads derived from real-world traces corresponding to scientific, engineering, and industrial domains.
2) We analyze the behavior of the autoscalers when faced with sudden peaks in resource demand by using bursty, real-world workloads (Section V).
3) We are the first to analyze the impact of the allocation policy on the performance of autoscalers (Section VI).
4) We are the first to investigate the behavior of autoscalers running across a diverse set of datacenter environments (Section VII). We specifically consider here the impact of datacenter utilization on autoscaling performance.

II. System Model

In this section, we describe a model of the datacenter environment considered in this work, focusing on autoscaling.

A. Model Overview

We consider in this work the following model of a cloud-based datacenter environment that uses an autoscaling component to take decisions dynamically about resource acquisition and release. As indicated by surveys of the field [5], [14], [15], the state-of-the-art includes many autoscalers, designed for different environments, workloads, and other design constraints. In contrast, our model matches the operations we have observed in practice in (i) *public cloud environments*, such as Amazon AWS and Microsoft Azure, (ii) *semi-public clouds* servicing industries with strict security and process restrictions, such as the datacenters operated by the Dutch company Solvinity for the financial industry and government public-services, and (iii) *private*, albeit *multi-institutional*, *clouds* such as the research-cloud DAS [16].

Figure 1 depicts an overview of our model. The workload is comprised of workflows, which are submitted by users to the *central queue* of the datacenter (component 1 in Figure 1). The transient set of machines available to users forms the *cloud* (5). A user-aware *scheduler* calls an *allocation policy* (3) to place the queued workflows onto machines available in the cloud. In parallel with the operations of the scheduler, the *resource monitor* (6) monitors periodically the utilization of each *active*, that is, allocated *machine*, and the state of the central queue. Starting from the monitoring information, and possibly also using historical data, the *autoscaler* component (7) periodically analyzes the demand (9) and the supply (10) of the dynamic system. Through the *resource manager* (4), the autoscaler can issue (de)allocation commands, to dynamically obtain or release resources according to the defined *provisioning policy* (8). Matching the state-of-the-art in such policies [4], [5], the provisioning policies we consider in this work are periodic and not event-based.

The periods used by the resource monitor and by the autoscaler evaluation, that is, the *monitoring interval* and the *autoscaling interval*, respectively, are adjusted through configuration files.

B. Workflows

A workflow is a set of tasks with precedence constraints between them. When all precedence constraints of a task have been fulfilled, it can be executed. The concept of a workflow fits both compute-intensive and data-intensive (i.e., for dataflows) tasks, and the precedence constraints frequently express data dependencies between tasks.

In this work, we use the common formalism of Directed Acyclic Graphs (DAGs) to model workflows. In this formalism, tasks are modeled as vertices, and precedence constraints are modeled as edges. DAGs are used across a variety of application domains, in science [9], industrial processes [10], and engineering [11]. DAGs also fit well web hosting [8] and multi-tier web applications [17]. For big data workloads, MapReduce workflows are frequently modeled as simple DAGs, where mappers and reducers are modeled as nodes, and the data dependencies between mappers and reducers are modeled as links in the graph; Spark-like dataflows are also common.

C. Provisioning Policies

We consider in this work two classes of autoscalers: (i) general autoscalers that are agnostic to workflows, and (ii) autoscalers that exploit knowledge and structure of workloads

[1]Taxonomy introduced by Kishor Trivedi at WEPPE'17.

TABLE I: The design and setup of our experiments. In bold, the distinguishing features of each experiment. (AS = autoscaling.)

ID	Experiment Focus (Section)	Input Workload	# System Clusters	# Resources per cluster	AS policies	Allocation Policies	Workflow Metrics	Autoscaler Metrics	System utilization
E1	Domain (§IV)	**T1, T2, T3**	50	70	All	FillworstFit	-	**All**	**70%**
E2	Bursty (§V)	**T2 (dupl), T4**	**Variable**	70	All	FillworstFit	**NSL**	-	70%
E3	Allocation (§VI)	T4	50	70	All	**All**	-	**Supply**	**Variable**
E4	Utilization (§VII)	T3	**Variable**	70	All	FillworstFit	NSL	All	**[10 – 90]%**

of workflows. All policies used in this work have already been used in prior studies in the community [4], [5] and provide a representative set of general and workflow-aware policies. They also cover a variety of strategies, based on different optimization goals such as throughput, (estimated) level of parallelism, and queue size.

We now describe the provisioning policies we study in this work, in turn:

1) Reg: employs a predictive component based on second-order regression to compute future load [18]. This policy works as follows: when the system is *underprovisioned*, that is, the capacity is lower than the load, *Reg* takes scale-up decisions through a reactive component that behaves similarly to the *React* policy (described in the following). When the system is *overprovisioned*, *Reg* takes scale-down decisions based on its predictive component, which in turn uses workload history to predict future demand.

2) Adapt: autonomously changes the number of virtual machines allocated to a service, based on both monitored load-changes and predictions of future load [19]. The predictions are based on observed changes in the request rate, that is, the slope of the workload demand curve. *Adapt* features a controller aiming to respond to sudden changes in demand, and to not release resources prematurely. The latter strategy aims to reduce oscillations in resource availability.

3) Hist: focuses on the dynamic demands of multi-tier internet applications [17]. This policy uses a queueing model to predict future arrival rates of requests based on hourly histograms of the past arrival-rates. *Hist* also features a reactive component that aims to handle sudden bursts in network requests, and to self-correct errors in the long-term predictions.

4) React Policy: takes resource provisioning decisions based on the number of active connections [20]. *React* first determines the number of resource instances below or above a given threshold. New instances are provisioned if all resource instances have a utilization rate above the threshold. If there are underutilized instances and at least one instance has no active connections (so, the instance is not in use), the idle instance is shutdown.

5) ConPaaS: is designed to scale web applications at fixed intervals, based on observed throughput [21]. Using several time series analysis techniques such as Linear Regression, Auto Regressive Moving Average (ARMA), and Multiple Regression, *ConPaaS* predicts future demand, and provisions resources accordingly.

6) Token: is designed specifically to autoscale for workflows [22]. By using structural information from the DAG structure, and by propagating execution tokens in the workflow, this policy estimates the level of parallelism and derives the (lack of) need for resources.

TABLE II: The four types of workloads and their characteristics. (W = number of workflows.)

ID	Source	Domain	W	# Tasks
T1	SPEC Cloud Group [5]	Scientific	200	13,876
T2	Chronos [10]	Industrial	1,024	3,072
T3	Askalon EE [11]	Engineering	757	45,786
T4	Askalon EE2 [11]	Engineering	3,551	122,105

7) Plan: predicts resource demand based on both workflow structure (as *Token* also does) and on task runtime estimation [5]. *Plan* constructs a partial execution plan for tasks that are running or waiting in the queue, considering task placement only for the next autoscaling interval. The resource estimation is derived by also considering the number of resources that already have tasks assigned to them; *Plan* places tasks on unassigned resources using a first-come, first-served order (FCFS) allocation policy.

III. Experimental Design

In this section we present our experimental design. Table I summarizes the design and setup of our experiments. For each experiment, the design considers deploying in turn the competing autoscalers (Section II-C) in a carefully controlled environment, subject to workloads that derive from real-world operation. Each experiment reports traditional and modern performance metrics.

A. Experiments

As indicated by Table I, our experimental design progresses systematically through experiments that test the impact on various performance metrics of the application domain, of the workload arrival process (and especially of *Bursty* arrivals), of the interplay of provisioning-allocation policies, and of the average system utilization. E1 runs in turn three workloads (explained in Section III-B) from distinct domains, using a setup derived from a state-of-the-art industrial setup and reporting all autoscaler metrics (explained in Section III-C2). E2 focuses on bursts in workload, which tests whether autoscalers can react to sudden changes in the task-arrival rates. E2 reports various workflow-related metrics (explained in Section III-C1). E3 measures the impact of allocation policies on the performance of autoscalers, using for this T4, the most stressful workload we use in our study. E3 reports a subset of the autoscaler metrics. E4 investigates the performance of autoscalers running of

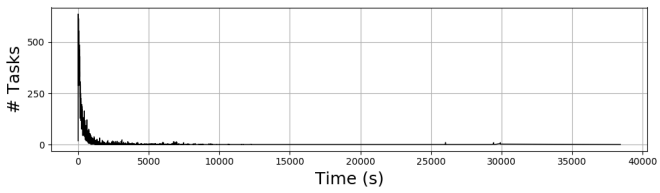

Fig. 2: The arrival rate of Askalon EE2 (T4).

systems with varying average resource utilization, using for this T3, a bursty and thus stressful workload but of a scale that allows us to conduct all experiments for all utilization levels. For more experiments, details, and complete results, see our technical report [23].

In all experimental setups, the autoscaling interval is set to 30 seconds for all experiments, which is a common setup in previous work, including [5]. The resource manager receives real-time updates of the resource utilization of each site. We do not include file transfer times or site boot-up times in any of the experiments. By default, the simulator schedules tasks using the `FillWorstFit` policy (the other policies introduced in Table I are discussed in Section VI). `FillWorstFit` selects the site with the most available resources and assigns as many tasks as possible, in a first-come-first-serve order, before selecting again. Matching prior work [5], clusters are only deallocated when they are idle, i.e., no tasks are being executed, but at least one cluster is kept running at all times.

Experiments E2 and E4 use a tailored setup, where the system is subjected to an arbitrary average utilization. We achieve this utilization by either scaling the infrastructure (results in Section VII) or the workload (results in our technical report [23]); scaling workloads derived from real-world traces has known methodological drawbacks and pitfalls, such as losing hidden characteristics and correlated behavior [24].

To scale the infrastructure while keeping its homogeneous structure, we compute the number of sites needed to achieve the target utilization. To this end, first, we compute per workflow its critical-path length (CP as defined in Section III-C1) and its *total load*, as the total amount of CPU seconds needed to complete all tasks of the workflow under ideal conditions. Second, we compute for the entire workload the *overall execution time* under ideal conditions, as the difference between the earliest submission time of all workflows, and the latest completion time when for each workflow its makespan is equal to CP (the ideal runtime), and the *total load* as the sum of total loads of each workflow. Last, from the overall execution time and the total load of the entire workload, we compute the number of clusters required to obtain the desired system utilization.

B. Workloads

In total, this work uses four different workloads, whose source, application domain, and characteristics are summarized in Table II. Each workload consists of multiple workflows. The required amount of CPUs and runtime per task are known a priori.

Workload T1 has been constructed by the SPEC RG Cloud Group to represent the scientific domain. All other workloads are real-world traces taken from either production environments or scientific clusters. T2 is derived from the Chronos production environment at Shell, and contains workflows whose structure matches a chain of three levels (tasks). T3 and T4 are traces from the Askalon cluster, corresponding to (chemical) engineering. Both feature a burst of workflow arrivals at the start of the workload, exemplified by Figure 2.

TABLE III: The metrics used in this work, grouped by level.

Workflow-level metrics	
M	Makespan
W	Wait time
R	Response time
NSL	Normalized schedule length
CP	Critical path length
Autoscaler-level metrics	
A_U	Underprovisioning accuracy
A_O	Overprovisioning accuracy
$\bar{A}_U$	Normalized underprovisioning accuracy
$\bar{A}_O$	Normalized overprovisioning accuracy
T_U	Time underprovisioned
T_O	Time overprovisioned
k	Average fraction of time of overprovisioning trends
k'	Average fraction of time of underprovisioning trends
M_U	Average number of idle resources
$\bar{V}$	Average number of resources
$\bar{h}$	Average accounted CPU hours per Virtual Machine (VM)
$\bar{C}$	Average charged CPU hours per VM

C. Metrics

In this performance study we use both traditional and modern metrics, which we divide in two categories: workflow-level and autoscaler-level metrics. Table III provides an overview of these metrics.

1) Workflow-level Metrics: quantify how well the system, autoscaler comprised, processes workflows. In this work we consider the following five workflow-level metrics.

For a workflow: The makespan (M) is the time elapsed between the start of its first task until the completion of its last task. The wait time (W) is the time elapsed between the arrival and the start of execution of a workflow's first task. The response time (R) is the total time the workflow stays in the system. It is the summation of the makespan and wait time of the workflow. The critical path length (CP) is the minimal duration of the workflow, under ideal conditions, considering all precedence constraints. The Normalized Schedule Length (NSL) is the response time, normalized by CP [25].

2) Autoscaler-level Metrics: are metrics quantifying the elasticity of systems that autoscale. We use here the modern elasticity metrics defined by the SPEC Cloud Group [26], which are used already in practice and in the scientific community [5].

In the SPEC Cloud Group's methodology, the *average demand* is the amount of resources that are required to uphold a Service Level Objective (SLO) at each autoscaling step, divided by the execution time. Similarly, the *average supply* is the average amount of resources provisioned over the execution time. In our model, we count as supplied any resource that is allocated to the user, so both in a running state or shutting down.

The SPEC methodology proposes various metrics capturing the accuracy of the elastic process. The *underprovisioning accuracy* metric (A_U) captures lacking resources, by summing the amount of resources that the autoscaler is not able to provide, relative to the momentary demand, normalized by the total amount of resources available during the experiment. Correspondingly, the *overprovisioning accuracy* metric (A_O) captures excessive provisioning.

When the amount of resources under- or over-provisioned can vary considerably over time, SPEC proposes to capture a more accurate representation of under- and overprovisioning, by normalizing the under- and overprovisioning accuracy metrics by the momentary resource demand and supply, respectively. Thus, we also use in our study the metrics *normalized overprovisioning accuracy* ($\bar{A}_O$) and *normalised accuracy underprovisioning* ($\bar{A}_U$).

The number of resources provisioned needs to follow the demand curve as accurately as possible. The *ideal autoscaler* provisions resources so that the system is executing all eligible tasks without having any idle resources, which is captured by the SPEC metric *average number of idle resources* (M_U). We also use in this work the SPEC metrics for quantifying if the system is constantly off the demand curve, or is off sporadically yet with large deviations: *time underprovisioned* (T_U) and *time over-provisioned* (T_O), which measure the fraction of time a system is under- or overprovisioned, respectively.

Last, we also use the metrics defined by the SPEC Cloud Group to capture resource consumption and cost: the *average amount of allocated resources* ($\bar{V}$), which provides insight into the costs of an autoscaler; the *average accounted CPU hours per VM* ($\bar{h}$), which measures the amount of hours a VM was used, that is, the effective CPU-hours used; and the *average charged CPU hours per VM* ($\bar{C}$), which measures the total amount of CPU hours a VM is charged and is dependent on the cost model used by the cloud provider (we use in our work the common per-hour charging that clouds such as Amazon EC2 use for some of their popular IaaS services).

D. Implementation Details

We implement the model from Section II as a simulation prototype in the OpenDC collaborative datacenter simulation project [27]; the resulting prototype, OpenDC.workflow, uses and extends significantly earlier code from the DGSim project [28].

The prototype is implemented in Python 2.7 and features all components highlighted in Section II-A. The implementation is modular, allowing components such as autoscaling and allocation policies to be swapped by configuration. The prototype is open-source and available at www.github.com/atlarge-research/opendc-autoscaling-prototype.

IV. Application-Domain Experiment

The aim of this experiment is to answer our research question *Does the application domain have an impact on the performance of autoscalers?*, by running workloads from distinct domains. In the state-of-the-art comparison study [5], all workloads are from the scientific domain and synthetically generated. By using real-world traces from different domains, we investigate if differences are observed running these workloads, expanding prior knowledge in this field.

Our main results in this section are:

- **We find significant differences between autoscalers, when scheduling for different application domains.**
- **The workload domain influences autoscaler under- and overprovisioning behavior significantly.**

A. Setup

In this experiment we use T1, T2, and T3, covering three distinct domains. Each workload consists of multiple distinct workflows, covering a range of applications from each domain. The Chronos workload is composed of workflows that process Internet-of-Things (IoT) sensor data, sampled periodically. The arrival pattern of this workload has an exponential pattern. Every minute 2^i workflows arrive, where $i \in [0-9]$ corresponds to the amount of minutes into the workload. As the sensors sample at a continuous rate, this workload can be repeated indefinitely. The Askalon EE workload originates from the engineering domain and features a huge initial burst. Roughly 16,000 tasks arrive in the first minute. Overall Askalon EE spans 49 minutes.

We use a tailored setup for each workload, where the number of clusters of the first setup is scaled to reach a system utilization of 70%, a representative number in supercomputing [29]. Each cluster contains 70 resources, matching the state-of-the-art industrial setup described in [10]. To measure the differences between autoscalers, we use the autoscaler-level metrics defined in Section III-C2.

B. Results

The results of this experiment are visible in Table IV. Overall, we observe significant differences for certain metrics per workload. Some autoscalers severely overprovision on a workload, while it may underprovision on another. This indicates that the aspects of the application domain, such as structure, arrival rate and complexity plays an important role on the performance of an autoscaler.

In particular, if we look at A_U, we notice that all autoscalers have similar values for all workloads. The $\bar{A}_O$, M_u, $\bar{V}$, $\bar{h}$, and $\bar{C}$ metrics for the Askalon EE workload are significantly different for Hist, ConPaaS and somewhat for Token, indicating these autoscalers waste resources significantly more than others on this particular workload. We ascribe these observations to the burst in Askalon EE. Hist and ConPaaS keep a history of past arrival rates, effectively biasing future

TABLE IV: The elasticity results, per workload, per autoscaler (AS), using a *workload-scaled* infrastructure.

AS	Workload	A_U	A_O	$\bar{A}_U$	$\bar{A}_O$	T_U	T_O	k	k'	M_U	$\bar{V}$	$\bar{h}$	$\bar{C}$
React	Chronos	70.7	17.6	27.8	17.6	45.9	50.5	0.0	0.0	36.0	202.3	3,467.9	21.0
	Askalon EE	211.5	16.1	32.4	47.6	40.0	56.0	4.0	1.0	16.7	1,319.9	2,121.3	58.9
	SPEC	54.8	38.9	15.7	38.9	33.7	64.4	1.6	0.8	70.8	68.8	3,538.9	784.0
ConPaaS	Chronos	70.7	17.6	27.8	17.6	45.9	50.5	0.0	0.0	36.0	202.3	3,467.9	21.0
	Askalon EE	211.5	50.4	31.4	55.8	39.0	60.0	7.0	0.0	51.0	2,088.1	3,355.9	93.2
	SPEC	52.9	40.2	14.9	40.2	32.9	66.0	2.6	0.0	72.5	70.0	3,597.6	797.0
Hist	Chronos	70.7	17.6	27.8	17.6	45.9	50.5	0.0	0.0	36.0	202.3	3,467.9	21.0
	Askalon EE	211.5	56.2	31.4	56.2	39.0	60.0	7.0	0.0	57.8	2,218.3	3,565.2	99.0
	SPEC	53.9	40.2	14.9	40.2	33.0	66.0	2.6	0.0	72.5	70.0	3,597.6	797.0
Adapt	Chronos	70.7	17.6	27.8	17.6	45.9	50.5	0.0	0.0	36.0	202.3	3,467.9	21.0
	Askalon EE	211.5	17.6	32.4	47.5	40.0	56.0	4.0	1.0	18.2	1,353.6	2,175.4	60.4
	SPEC	54.9	38.8	15.7	38.8	33.6	64.6	1.6	0.8	70.9	68.8	3,538.9	784.0
Plan	Chronos	70.7	17.7	27.8	17.7	45.9	50.5	0.0	0.0	36.0	202.3	3,467.9	21.0
	Askalon EE	210.0	17.8	33.1	45.9	40.7	53.6	2.0	2.0	18.4	1,350.0	2,169.6	60.7
	SPEC	54.2	38.1	15.7	38.1	33.7	63.6	0.9	0.9	70.0	68.2	3,507.3	777.0
Reg	Chronos	70.7	17.6	27.8	17.6	45.9	50.5	0.0	0.0	36.0	202.3	3,467.9	21.0
	Askalon EE	210.0	16.1	33.1	45.6	40.7	53.6	2.0	2.0	16.7	1,312.4	2,109.2	59.0
	SPEC	53.9	39.7	15.4	39.7	33.4	65.3	2.4	0.5	71.6	69.4	3,570.5	791.0
Token	Chronos	70.7	17.6	27.8	17.6	45.9	50.5	0.0	0.0	36.0	202.3	3,467.9	21.0
	Askalon EE	211.5	16.8	31.4	51.4	39.0	60.0	7.0	0.0	17.4	1,335.3	2,146.1	59.6
	SPEC	53.4	40.3	14.8	40.3	32.7	66.1	2.6	0.0	72.5	70.0	3,597.6	797.0

predictions due to this one time peak, while Token seems to overestimate the level of parallelism.

Overall, our results suggests the aspects that come with an application domain, such as structure and complexity, play an important role on the performance of an autoscaler. This indicates the choice of autoscaler is non-trivial for a given application domain and should be carefully benchmarked.

V. Bursty Workload Experiment

Bursts are common in cloud environments [19]. The sudden increase in demand of resources require a proactive approach to ensure task execution is not delayed. The goal of this experiment is to provide insights to answer *How well can autoscalers handle a significant burst in arriving tasks?*.

Our main results in this section are:

- **Workload-agnostic autoscalers perform equally well compared to workload-specific autoscalers**.
- **The Plan autoscaler creates an order of magnitude more task delay than the other autoscalers when executing the Chronos workload.**

A. Setup

To investigate the impact of bursts, we use two real-world workflows from two distinct domains.

The first workload is EE2 from the Askalon traces. EE2 features a one-time burst at the start of the workload, visible in Figure 2. In the first minute, around 24,000 tasks arrive which is not an uncommon amount in grids and clusters [11].

The second workload is a scaled version of the industrial Chronos workload used in Section IV, visible in Figure 3.

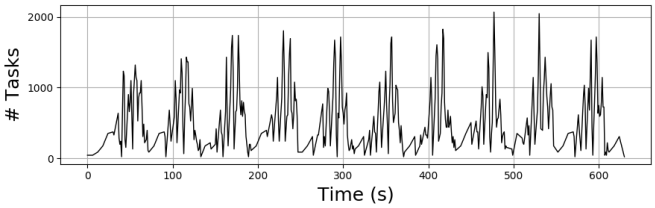

Fig. 3: The arrival rate of Chronos (duplicated 22 times).

The workload is scaled so that the highest peak – on a minute basis – matches that of the Askalon EE2 trace. This workload remains representative as the workload scales linearly with the amount of IoT sensors applied in Shell's Chronos infrastructure.

The infrastructure is designed to have an average of 70% resource utilization, for the reasons stated in Section IV. To measure the impact of bursts, we measure the cumulative delay, NSL, and M per autoscaler, per workload. We compute the NSL using the workload makespan and CP. The delay is computed by subtracting the CP of a workload's makespan. By using these metrics, we investigate if there are differences in how autoscalers handle burstiness.

B. Results

To observe the impact of these bursts per autoscaler, we measure the M per workload and compute the overall NSL, per autoscaler. We only visualize the results for the Chronos workload, which shows the most variation. More results can be found in our technical report [23].

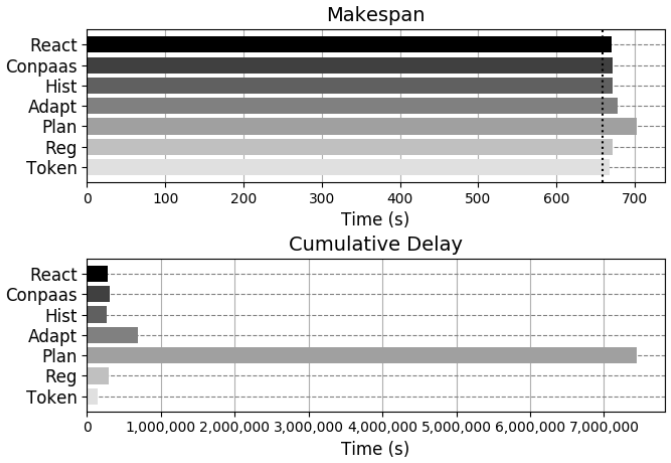

Fig. 4: The makespan and cumulative delay per autoscaler for the Chronos workload. Color coding matches Figure 6.

TABLE V: The NSL, per autoscaler, per workload.

Workflow	React	ConPaaS	Hist	Adapt	Plan	Reg	Token
Askalon EE2	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Chronos	1.02	1.02	1.02	1.03	1.07	1.02	1.01

The results for Chronos are visible in Figure 4. This top figure shows per autoscaler the M. The CP of the workload is annotated by a black dotted line. From this figure we observe the Plan autoscaler has a slightly higher M than the other autoscalers. From the M and CP we compute the NSL for each combination, see Table V. From this table we observe little difference between the autoscalers for both workloads.

The bottom figure shows the cumulative delay, per autoscaler for the Chronos workload. From this figure we observe that the Plan autoscaler creates an order of magnitude more task delay than the other autoscalers. This is in contrast with the NSL being roughly equal to the other autoscalers. We therefore conclude that while the workload finishes roughly in the same time compared to the other autoscalers, during execution tasks are delayed significantly more when using Plan. Plan allocates considerably fewer clusters to process tasks compared to the other autoscalers, causing significant task delay. This would likely lead to several QoS violations.

Overall, workload-agnostic autoscalers perform similarly in terms of NSL and might even outperform worklow-specific autoscalers regarding task delay.

VI. Different Allocation Policies Experiment

Besides the availability of resources, other components can affect the performance of a system. The goal of this experiment is to answer the research question *Does the choice of allocation policy have an impact on the performance of the autoscalers?*, i.e. investigate and quantify the impact of the allocation policy on the performance of the autoscalers.

Our main results in this section are:

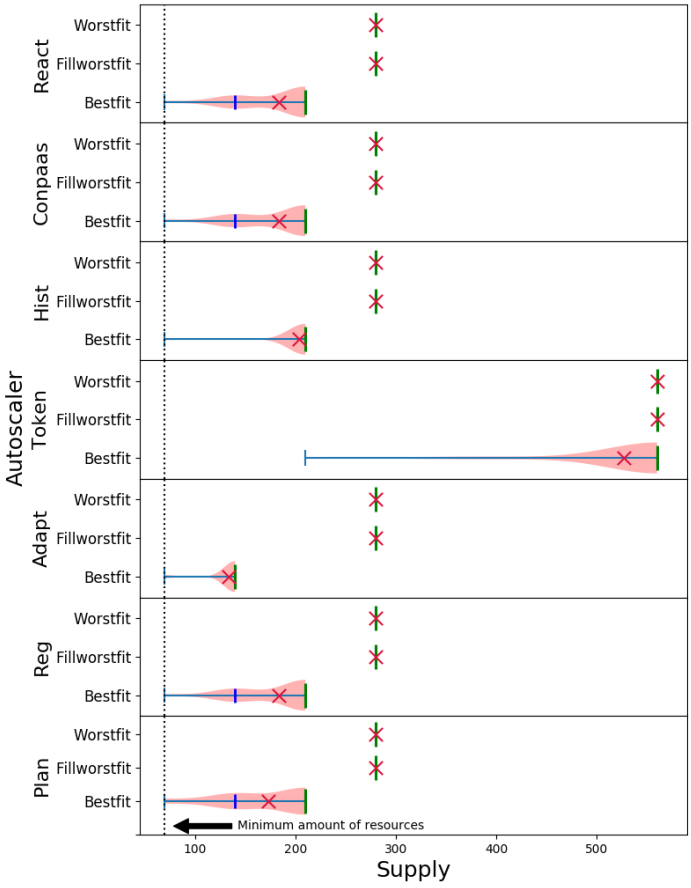

Fig. 5: The supply in resources, per allocation policy, per provisioning policy. (For each allocation-autoscaler pair, the combined violin and box-and-whiskers plots summarize the empirical distribution of observed supply.)

- **The allocation policy has a direct influence on the behavior of all autoscalers.**
- **The interplay between deallocation strategy and allocation policy can have a significant impact on the average supply in a system.**

A. Setup

In this experiment we run all possible combinations of allocation policies and autoscalers using a state-of-the-art industrial setup and workload [10].

To measure the impact of allocation policies on the behavior of the autoscalers, we have implemented three different policies in our simulator: FillWorstFit, WorstFit, and BestFit. WorstFit selects the site with the most available resources. After assigning a task it re-evaluates which site has the most available resources. BestFit selects per task the site with the least available resources where the task still fits. None of these policies use a greedy backfilling approach.

To measure the impact of proposed allocation policies, we measure the supply of resources throughout the experiment.

B. Results

The results of this experiment are visible in Figure 5. On the vertical axis we show seven groups, one for each autoscaler. On the horizontal axis, we depict the distribution of supply. For each autoscaler and allocation policy combination, we show the distribution of the supply choices. The minimum and maximum as well as the 25th and 75th percentiles are denoted by blue bars. The median is denoted by a larger green bar. The distribution of supply is visualized by a shaded area around the range. The average supply is denoted by a red cross symbol.

Overall we observe some differences between the autoscalers. From the perspective of the autoscalers, besides Token and Adapt, all autoscalers have the same minimum and maximum supply for BestFit. And besides Token, an equal amount of supply for WorstFit and FillWorstFit.

From the perspective of the allocation policies, we can observe the BestFit allocation policy has a significant lower average supply than the other two policies, for every autoscaler. As FillWorstFit and WorstFit assign jobs to the most idle clusters, clusters will rarely be considered idle and thus will not be deallocated. One way to resolve this scenario is to migrate tasks by e.g. migrating the VM or interrupt and reschedule tasks. This is part of our ongoing work. Additionally, we observe differences in the distribution and average of supply per autoscaler for BestFit.

From Figure 5, we observe that Adapt has the lowest maximum allocated resources, which impacts the response time of the workload. All setups running FillWorstFit and WorstFit require 650 seconds to process the workload. Token running BestFit also requires 650 seconds as it provisions more machines. Adapt running BestFit requires 694 seconds due to underprovisioning. All other autoscalers running BestFit require 654 seconds.

From these observations we conclude that the allocation policy can have a direct influence on the behavior and performance of autoscalers. As a result, comparing autoscaler can only be done fairly when using the same allocation policies.

VII. Different utilization experiment

In the current state-of-the-art [5], the system utilization is 39.5% for workload T1, assuming all VMs were allocated throughout the experiment. Prior work demonstrates a resource utilization of 70% is possible [29] in supercomputing. In this experiment, we shed light on our research question *How do autoscalers behave when faced with different resource environments?*. To investigate how autoscalers operate when faced with different amount of resources, we scale the infrastructure based on a target resource utilization based on the amount of CPU seconds a workload contains. We compare the normalized under- and overprovisioning for each autoscaler.

Our main results in this section are:

- **The fraction of time of overprovisioning differs per autoscaler significantly at lower utilization, yet converges for higher utilization.**
- **Autoscalers that significantly overprovision more resources do not yield a better NSL.**

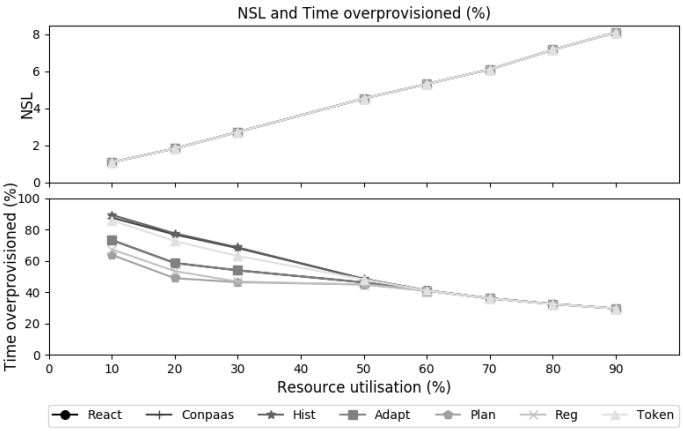

Fig. 6: The NSL and $\bar{A_O}$ per utilization per autoscaler. All curves overlap in the top part of the figure.

A. Setup

In this experiment we run the Askalon EE workload, a workload from the engineering domain. This workload contains a load of 2,823,758 CPU seconds and spans in total 2,998 seconds, based on the submission times and critical paths of the workflows. We vary the number of clusters to obtain 10%, 20%, ..., 90% resource utilization. Each cluster has 70 resources, for reasons described in Section IV. Note that since autoscalers predict the amount of VMs required, the system may allocate more VMs since the resource manager computes the required amount of clusters (ceiled) to meet the prediction (up to the maximum amount of clusters specified).

We measure the normalized under- and overprovisioning time in the system and NSL of processing the workload. This provides us with insight into the trade-offs autoscalers make when allocating resources.

B. Results

The results are visible Figure 6, where we plot for each resource utilization percentage the $\bar{A_O}$ and NSL. $\bar{A_O}$ shows significant differences between the autoscalers. We expect autoscalers to somewhat overprovision, as resources are incremented per 70. However, we observe Hist, ConPaaS, and Token overprovision significantly more than the other autoscalers, while yielding no benefit in NSL. We ascribe this behavior for Hist and ConPaaS to keeping a history of incoming jobs. For EE, roughly 16,000 jobs arrive at the start, causing the histogram to be biased towards overprovisioning resources. For Token, the level of parallelism influences the amount of resources provisioned. Token estimates a high level of parallelism for the EE workload, causing overprovisioning. React and Plan perform the best as they feature the lowest overprovisioning time while having a similar NSL compared to the other autoscalers.

From these observations we conclude that at lower system utilization, some autoscalers overprovision more than others, yet converges for high system utilization. The autoscalers that overprovision more resources do not yield a better NSL.

VIII. Threats to Validity

We discuss in this section the threats and limitations to the validity of this work we have identified, and the remedial steps we have taken. We see the use of a simulator, instead of a real-world setup, as the most important threat to the validity of our work. To address this threat, we have validated our simulator (i) manually, using small example workflows, (ii) automatically, generating workflows and testing the output matches Little's Law, (iii) re-running the experiments defined in [5], for which we have real-world results, and (iv) using as input workload for all experiments real-world traces to further improve the resemblance of our simulations to the real world. The alternative of using a real-world setup would not have been possible without considerable expense and delay.

Albeit considerably more diverse than the experimental setups used in previous studies, our simulation-based constructs still suffer from limitations: the simulated infrastructure is homogeneous, the workflows are compute-intensive, etc. The simulated infrastructure used in this work matches many current private/hybrid cloud setups, which are often homogeneous [30] or provide a homogeneous view of virtualized resources [31]. The main challenge of taking heterogeneity into account, expresses itself through different processing speed per resource.

We simulate compute-intensive workflows; in reality, data-intensive workflows are becoming more common, from scientific workflows [32], [33], to MapReduce [34] and other big data workflows. However, data-intensive workflows also involve resource management and scheduling besides autoscalers, for example, caching [35] and auto-tiering techniques, whose study jointly with autoscalers greatly exceeds the scope of this work.

The internal validity of our study is affected by the choice of using a single exemplary trace for each of the scientific, engineering, and industrial domains. Although more could in general be better, currently the community does not have a set of traces with provably good coverage or representation for any of these domains; in other words, there is no Parallel or Grid Workloads Archive for workflows that the community can use for provably representative experiments. Establishing such an archive goes beyond the scope of this work.

The external validity of our study has as main threat generalizing beyond the findings of this study. The domains used in this work do not represent the full spectrum found in cloud environments, for example, workloads from health, financial [36], and business domains.

IX. Related Work

This study complements, and by design of research questions significantly extends, the large body of related work in the field surveyed by Lorido-Botran et al. [15] and by Vaquero et al. [14]. Beyond the new research questions we propose, our study is the first to use an experimental approach with workloads from multiple domains, and diverse datacenter environments.

Closest to our work, Ilyushkin et al. [5] conduct the first systematic comparison of autoscalers for workflows, using real-world experimentation. The experimental design is limited: their work focuses on the scientific domain, and, due to the use of real-world resources, limits the amount of machines to 50 and the resources used per task to a single core.

Also close to this work, Papadopoulos et al. [4] introduce a performance evaluation framework for autoscaling strategies in cloud applications (PEAS). PEAS measures the performance of autoscaling strategies using scenario theory. Similarly to this work, they use a simulation-based experimental environment, analyzing the elasticity of six autoscaling policies for workload traces collected from the web-hosting domain. In contrast, this work focuses on workloads of *workflows*, leading to significantly new scheduling constraints, on different application domains, and on different types of metrics (in particular, also workflow-level).

Our work also complements the specialized performance comparisons presented by authors of new autoscalers, such as the work of Han et al. [37] (domain-agnostic), Hasan et al. [38] (domain-agnostic), Heinze et al. [2] (stream-processing domain), Mao et al. [39] (scientific workflows), Jiang et al. [40] (web-hosting domain), Dougherty et al. [3] (focus on energy-related metrics). In contrast to these studies, ours focuses on deeper analysis focusing on new research questions, on different and more diverse application domains, etc.

X. Conclusion and Ongoing Work

Autoscaling, the process of acquiring and releasing resources at runtime, is an important, non-trivial task at the core of datacenter operations. To help with understanding how autoscalers work, prior work has performed systematic analysis and comparisons, yet the results still lack in the diversity of application domain, choice of metrics, and environments. Addressing this lack of diversity, in this work we perform a comprehensive comparison of state-of-the-art autoscalers.

We ask new research questions about the operational laws of autoscalers managing workloads of workflows. To answer these questions, we conduct trace-based simulations, measuring the impact of autoscaling across a variety of datacenter environments, workloads, and metrics. We use workloads from three different domains, scientific, industrial processes, and engineering. We analyze the performance effects of workload burstiness, of the interplay between allocation and autoscaling, and of the level of utilization in the datacenter.

Our study gives strong, quantitative evidence about autoscaling performance, including findings such as:

- The application domain has a significant impact on the performance of an autoscaler.
- For bursty workloads, workload-agnostic and workfload-specific autoscalers result in similar NSL performance.
- The allocation policy has a direct impact on autoscaling performance. The pair allocation and provisioning policy should be co-designed and considered together when deploying systems.

- Especially for lower system utilization, the Hist, ConPaaS, and Token autoscalers overprovision more than the others, while yielding no better NSL.

In our ongoing work, we will study the impact of heterogeneity in datacenters on autoscalers, measure the impact of different application domains, using traditional and emerging cost metrics (e.g., the finer-grained cost-models released for selected resources by Microsoft and Google in mid-2017) to compare autoscalers, include boot-up and network transfer times, etc. We have also indicated throughout this work several directions for future work, for example, investigating the effect of migrating jobs on the size clusters, e.g., deallocating entire clusters when not used, to gain further insight into the effect of using different allocation policies.

ACKNOWLEDGMENTS

This work is supported by the Dutch projects Vidi MagnaData and KIEM KIESA, by the Dutch Commit and the Commit project Commissioner, and by generous donations from Oracle Labs, USA.

REFERENCES

[1] Vasan *et al.*, "Worth their watts?-an empirical study of datacenter servers," in *HPCA*, 2010.
[2] Heinze *et al.*, "Auto-scaling techniques for elastic data stream processing," in *ICDE Workshops*, 2014.
[3] Dougherty *et al.*, "Model-driven auto-scaling of green cloud computing infrastructure," *FGCS*, 2012.
[4] Papadopoulos *et al.*, "PEAS: A performance evaluation framework for auto-scaling strategies in cloud applications," *TOMPECS*, 2016.
[5] Ilyushkin *et al.*, "An Experimental Performance Evaluation of Autoscaling Policies for Complex Workflows," in *ICPE*, 2017.
[6] Iosup and Epema, "Grid computing workloads," *IEEE Internet Computing*, vol. 15, no. 2, pp. 19–26, 2011. [Online]. Available: http://dx.doi.org/10.1109/MIC.2010.130
[7] Li, "Realistic workload modeling and its performance impacts in large-scale escience grids," *IEEE TPDS*, vol. 21, no. 4, pp. 480–493, 2010.
[8] Wu *et al.*, "Workflow scheduling in cloud: a survey," *TJS*, 2015.
[9] Juve *et al.*, "Characterizing and profiling scientific workflows," *FGCS*, vol. 29, no. 3, pp. 682–692, 2013.
[10] Ma *et al.*, "ANANKE: a Q-Learning-Based Portfolio Scheduler for Complex Industrial Workflows," in *ICAC*, 2017.
[11] Iosup *et al.*, "The Grid Workloads Archive," *FGCS*, 2008.
[12] Rodriguez and Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments," *CCPE*, vol. 29, no. 8, 2017.
[13] A. Iosup *et al.*, "Massivizing computer systems: a vision to understand, design, and engineer computer ecosystems through and beyond modern distributed systems," *CoRR*, vol. abs/1802.05465, 2018.
[14] Vaquero *et al.*, "Dynamically scaling applications in the cloud," *SIGCOMM*, 2011.
[15] Lorido-Botran *et al.*, "A review of auto-scaling techniques for elastic applications in cloud environments," *JGC*, 2014.
[16] Bal *et al.*, "A medium-scale distributed system for computer science research: Infrastructure for the long term," *Computer*, 2016.
[17] Urgaonkar *et al.*, "Dynamic provisioning of multi-tier internet applications," in *ICAC*, 2005.
[18] Iqbal *et al.*, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *FGCS*, 2011.
[19] Ali-ElDin *et al.*, "Measuring cloud workload burstiness," in *UCC*, 2014.
[20] Chieu *et al.*, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in *ICEBE*, 2009.
[21] Fernandez *et al.*, "Autoscaling web applications in heterogeneous cloud infrastructures," in *IC2E*, 2014.
[22] Ilyushkin *et al.*, "Scheduling workloads of workflows with unknown task runtimes," in *CCGRID*, 2015, pp. 606–616.
[23] Versluis *et al.* (2017) Technical report: A trace-based performance study of autoscaling workloads of workflows in datacenters. [Online]. Available: https://atlarge-research.com/lfdversluis/2017-11-24_lfdversluis_autoscaling-comparison.pdf
[24] Frachtenberg and Feitelson, "Pitfalls in parallel job scheduling evaluation," in *JSSPP*, 2005, pp. 257–282.
[25] Kwok *et al.*, "Benchmarking the task graph scheduling algorithms," in *IPPS/SPDP*, 1998.
[26] Herbst *et al.*, "Ready for rain? A view from SPEC research on the future of cloud metrics," *CoRR*, vol. abs/1604.03470, 2016.
[27] Iosup *et al.*, "The opendc vision: Towards collaborative datacenter simulation and exploration for everybody," in *ISPDC*, 2017.
[28] ——, "DGSim: Comparing grid resource management architectures through trace-based simulation," in *ECPP*, 2008.
[29] Jones *et al.*, "Scheduling for parallel supercomputing: A historical perspective of achievable utilization," in *JSSPP*, 1999.
[30] Ghanbari *et al.*, "Feedback-based optimization of a private cloud," *FGCS*, 2012.
[31] Sotomayor *et al.*, "Virtual infrastructure management in private and hybrid clouds," *Internet*, 2009.
[32] Eom *et al.*, "Speed vs. accuracy in simulation for i/o-intensive applications," in *IPDPS*, 2000.
[33] May *et al.*, "ZIB structure prediction pipeline: composing a complex biological workflow through web services," *Euro-Par*, 2006.
[34] Ousterhout *et al.*, "Making Sense of Performance in Data Analytics Frameworks." in *NSDI*, 2015.
[35] Uta *et al.*, "Towards resource disaggregation-memory scavenging for scientific workloads," in *CLUSTER*, 2016.
[36] Garrison *et al.*, "Success factors for deploying cloud computing," *CACM*, 2012.
[37] Han *et al.*, "Lightweight resource scaling for cloud applications," in *CCGrid*, 2012.
[38] Hasan *et al.*, "Integrated and autonomic cloud resource scaling," in *NOMS*, 2012.
[39] Mao *et al.*, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *SC*, 2011.
[40] Jiang *et al.*, "Optimal cloud resource auto-scaling for web applications," in *CCGrid*, 2013.