CEDULE+: Resource Management for Burstable Cloud Instances Using Predictive Analytics

Riccardo Pinciroli, Ahsan Ali, Feng Yan, and Evgenia Smirni

***Abstract*—Nearly all principal cloud providers now provide *burstable instances* in their offerings. The main attraction of this type of instance is that it can boost its performance for a limited time to cope with workload variations. Although burstable instances are widely adopted, it is not clear how to efficiently manage them to avoid waste of resources. In this paper, we use predictive data analytics to optimize the management of burstable instances. We design CEDULE+, a data-driven framework that enables efficient resource management for burstable cloud instances by analyzing the system workload and latency data. CEDULE+ selects the most profitable instance type to process incoming requests and controls CPU, I/O, and network usage to minimize the resource waste without violating Service Level Objectives (SLOs). CEDULE+ uses lightweight profiling and quantile regression to build a data-driven prediction model that estimates system performance for all combinations of instance type, resource type, and system workload. CEDULE+ is evaluated on Amazon EC2, and its efficiency and high accuracy are assessed through real-case scenarios. CEDULE+ predicts application latency with errors less than 10%, extends the maximum performance period of a burstable instance up to 2.4 times, and decreases deployment costs by more than 50%.**

***Index Terms*—Burstable instance, Cloud, Scheduling, AWS, Credit depletion period, Resource credit, Data-driven predictive analytics.**

I. Introduction

Most major cloud providers offer *burstable instances* to enhance the utilization of spare resources. Burstable instances can operate in *standard* or *unlimited* mode. In a newly-launched burstable instance that operates in standard mode, each resource (i.e., CPU, network, I/O) has initial credits that are used to boost performance beyond a *baseline* level. An instance can operate above baseline until it exhausts its initial credits, after that point performance stabilizes at baseline. During this *Credit Depletion Period* the credit consumption rate is a function of application resource usage, i.e., the higher the resource usage, the shorter the credit depletion period. For example, an AWS T2 instance consumes one CPU credit in a minute if its CPU is 100% utilized or in two minutes if CPU utilization is 50%. After the baseline level is reached, the accrual/consumption of credits reaches an equilibrium. An unlimited burstable instance can always work with maximum performance. The user is charged with further monetary cost only if the average resource utilization over 24 hours is higher than the baseline. Unlimited burstable instances are out of the scope of this paper and are not further considered.

An application enjoys bursting performance when served during the credit depletion period. Provided that the application service level objective (SLO) conditions are met, it is possible to extend the credit depletion period by carefully throttling instance resource usage [28]. When the credit depletion period ends, users may opt to move their application to a new instance (with new initial credits), perhaps even explore whether a different instance type may be better suited.

Although extending the credit depletion period by throttling resource usage is attractive, it faces multiple challenges: *i)* resource credit dynamics are not always clear and publicly available; *ii)* the search space is combination of instance types, different resource(s) and their limiting value(s), workload intensity: exhaustive application profiling within this vast space is prohibitively expensive; *iii)* as applications typically require several resources, limiting the usage of a single resource may affect the usage of other resources, for example, throttling network bandwidth may also reduce I/O throughput and vice versa. Supporting various application service level objectives while facing the above challenges is difficult.

This paper introduces a framework called *CEDULE uP-grade for muLti-instance bUrSts* (CEDULE+) that aims to improve system efficiency by throttling resource usage of multiple resources and cope with the aforementioned challenges using predictive analytics and a black-box approach (i.e., an application is characterized only by its workload and latency). CEDULE+ extends the credit depletion period of an instance while complying with the user-defined SLO (expressed as latency percentiles). It monitors the system to deploy the application on the burstable instance type that allows for the highest cost reduction, and it predicts the credit depletion period length to efficiently schedule instance migration before observing performance deterioration. CEDULE+ profiles applications in a lightweight manner by sparsely sampling their percentile latencies on distinct types of burstable instances (e.g., *t2.micro*, *t2.medium*, *t2.large*) and under different resource limitations. CEDULE+ employs an analytical model driven by quantile regression to predict the percentile latencies of applications under situations that are not profiled. Then, it selects the best combination of instance type, limitation type, and limiting value that complies with the user-defined SLO and minimizes costs. The framework integrates single resource throttling in [28, 35] to provide a multi-resource throttling solution with the additional consideration of multiple instance types and

R. Pinciroli is with the Computer Science Department, Gran Sasso Science Institute, Italy.
E-mail: riccardo.pinciroli@gssi.it
A. Ali and F. Yan are with the Computer Science Engineering Department, University of Nevada, Reno, Reno, NV 89557.
E-mail: {aali,fyan}@nevada.unr.edu
E. Smirni is with the Computer Science Department, William and Mary, Williamsburg, VA 23185.
E-mail: esmirni@cs.wm.edu

potential job co-location.

CEDULE+ is prototyped atop Amazon EC2. Three applications with different profiles of resource requirement are used for its evaluation: TPC-W [1], FTP, and Ceph [4]. TPC-W is a transactional web server benchmark whose CPU-intensive workload simulates the visits and queries to an e-commerce application. FTP is a protocol to download files between machines and works using the client/server paradigm. It is a network-intensive application and it uses network and I/O if the file to be downloaded is too large to be placed into the cache. Ceph is a platform that implements efficient and reliable distributed object storage. A Ceph cluster requires network, CPU, and I/O to process requests. These applications allow evaluating CEDULE+ on a variety of workloads. Results show that CEDULE+ can extend the credit depletion period of an instance up to 2.4 times and reduce costs by more than 50%, while complying with user-defined SLOs.

Major contributions of this paper are summarized below:

- We devise a lightweight data-driven strategy that quickly assembles application profiles within a huge search space.
- We formulate an optimization problem to minimize the monetary cost of applications by selecting the least expensive burstable instance type and, at the same time, maximize the credit efficiency of an instance by throttling the usage of CPU, network, and/or I/O.
- We propose an analytic model that accurately predicts the credit depletion period of an instance to proactively migrate the application before SLOs are violated.

II. Background

Here, we describe the credit mechanism in burstable instances and how to throttle resources.

A. Credit Mechanism

In burstable instances, CPU, network, and I/O usage are governed by a credit mechanism that allows each resource to boost up its performance during the credit depletion period. CPU credits allow a temporal burst of higher CPU utilization, e.g., one CPU credit allows boosting up the CPU utilization up to ten times for one minute. Each network credit allows the instance to transfer a certain amount of data over the network in a given period, e.g., a network credit provides 1 Mbps network bandwidth. I/O credits enable faster read/write operations per second, e.g., an I/O credit enables the execution of one I/O operation per second up to a maximum I/O bandwidth of 3000 operations per second [47]. There are three key components in the credit mechanism:

- *Initial credits.* A newly-launched burstable instance receives initial credits for each resource, e.g., a *t2.micro* instance gets 30 credits for CPU, 123,617 credits for network, and 5,400,000 credits for I/O.
- *Credit accrual.* An instance gains credits when it operates below baseline performance and the credit accrual rate depends on the resource usage – the lower the usage, the higher the accrual rate. For example, if CPU utilization is 10% below the baseline usage, it receives 1 credit every 10 minutes. The credit accrual is capped, e.g., the CPU of a *t2.micro* instance cannot receive more than 144 credits.
- *Credit depletion.* When an instance operates over baseline performance, it consumes credits at a rate that is determined by resource usage – the higher the usage, the faster credit depletes. For example, if CPU utilization is two times higher than the baseline CPU utilization, it depletes twice as fast the accrued credits.

When an instance operates at the baseline level, an equilibrium is reached between credit accrual and credit depletion, i.e., credits do not change. When an instance exhausts its resource credits, the system reverts to baseline performance level.

B. Throttling Resources: CPU, Network, and I/O

Previous work [27] shows that it is possible to extend the CPU credit depletion period by limiting CPU usage. There are several strategies for limiting the CPU usage of a process. *Nice* [61] changes the process priority and is useful for batch processing. *Cgroups* [52] is a Linux kernel feature that enables CPU usage limitation by controlling the amount of time that a process spends in CPU (called *quota*) over a specific *period*. Both parameters (i.e., *quota* and *period*) are specified by the user. *Cpulimit* [58] pauses a process execution to limit its CPU utilization in order to keep it under a defined value. *Cpulimit* requires the process ID and the maximum CPU usage allowed for that process as input parameters. For example, *cpulimit* with 50% limitation forces the CPU usage to be no higher than 50%. In this paper, we use *cpulimit* as the preferred tool to throttle CPU.

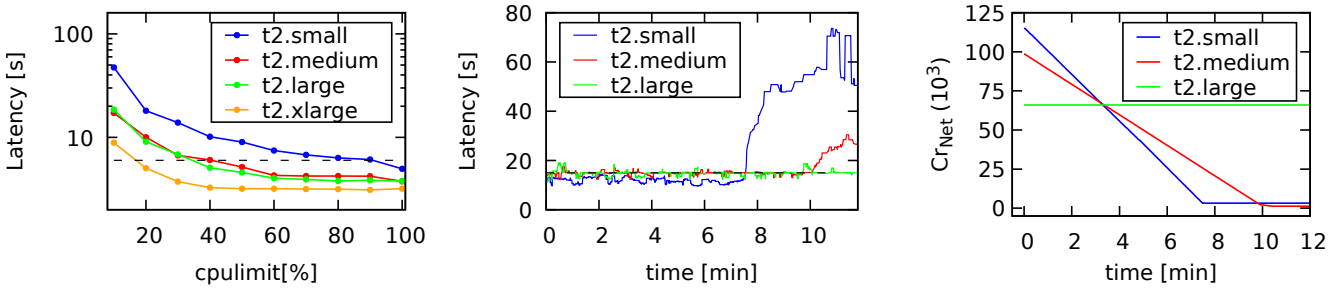
Throttling the bandwidth of an instance reduces the network credit consumption rate. Several traffic shaping tools allow delaying communication over the network. *Linux Traffic Control* [63] is a Linux kernel feature that allows specifying different network-related parameters, such as transmission rate and packet scheduling. *Wonder Shaper* [55] works with *Linux Traffic Control* and allows the user to limit network bandwidth by specifying a maximum amount of bits that can be transferred every second. Other tools include *Trickle* [3], *TrafficToll* [51], and *cgroups*. *WonderShaper* is the tool used in this paper for throttling network bandwidth.

I/O throttling can be achieved through *cgroups*, the tool used in this paper. It allows the user to specify the rates of read and write operations (i.e., read/s and write/s, respectively) performed on a specific I/O device. Their sum defines the number of input/output operations that can be executed every second (IOPS) on the device.

C. Instance Type and Throttling Performance

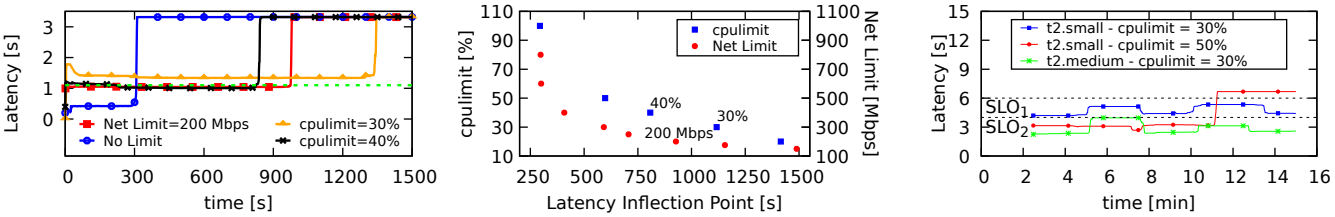
Different instance types provide different amounts of initial credits (i.e., different credit depletion periods) and distinct baseline performance. Note that only CPU and network performance is affected by the instance type. I/O performance depends only on the capacity of the I/O device [47].

Fig. 1 shows the performance of TPC-W and FTP when they are deployed on different instance types. Fig. 1(a) depicts the 95th-percentile latency of TPC-W (y-axis is in log-scale) as a function of *cpulimit*. As expected, more powerful instances



(a) 95th-percentile latency vs. *cpulimit*, TPC-W (b) 95th-percentile latency vs. time, FTP (c) Network credit depletion, FTP

Fig. 1: Performance of TPC-W and FTP when deployed on instances with different capacity.



(a) 95th-percentile latency (b) Limits vs. Latency inflection point (c) 95th-percentile latency

Fig. 2: Effect of resource throttling and instance upgrade on a Ceph storage system. Only one resource is throttled at a time.

allow the system to process the incoming requests with shorter latency. TPC-W complies with the SLO (see the dashed line) independently of the instance hosting the application. Different CPU limitations have different effects. For example, TPC-W on *t2.small* can comply with the SLO only if *cpulimit* > 90%. When the same application is deployed to *t2.xlarge*, the minimum *cpulimit* to satisfy the SLO is 20%. Fig. 1(b) shows the 95th-percentile latency of an FTP server, as a function of time, when 55 users concurrently download a file and the network bandwidth is throttled to 500 Mbps. The more powerful the instance, the longer the application can burst its performance since the credit depletion period (the latency inflection point) depends on the instance types. Fig. 1(c) depicts the network credit depletion as a function of time, across three instances. *t2.small* is the first one to exhaust all network credits, while *t2.large* does not experience any credit depletion due to limited resource requirements of the FTP application.

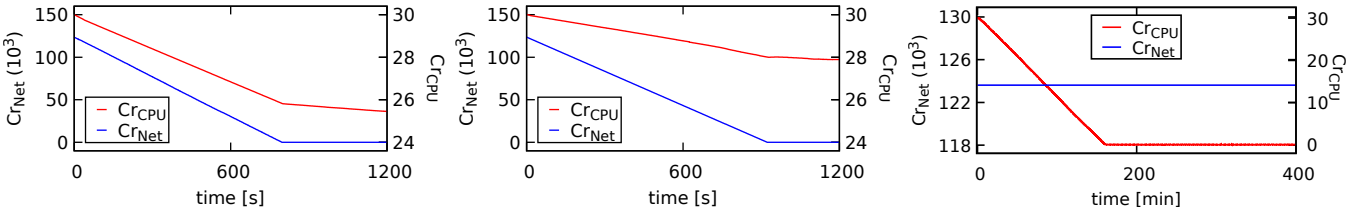
III. Throttling Challenges

We first look into the duration of the credit depletion period. The credit depletion period may be extended by throttling resource usage. This is shown in Fig. 2(a) that depicts the 95th-percentile latency of Ceph when network and CPU limitations are separately applied. If no resource is throttled, then the application can use all the available credits to process incoming requests. Initially, system performance is below the specified SLO (see the dashed line), but the credit depletion period ends after 300 seconds and the SLO is violated. Setting either the network bandwidth to 200 Mbps or the CPU utilization to 40% provides comparable latency. Although system performance is worse than without any limitation, the latency satisfies the SLO for a longer period (i.e., up to 900 seconds). Throttling CPU utilization even more (by setting *cpulimit*=30%) makes the credit depletion period longer than 1200 seconds. In this case, system performance violates the SLO.

Fig. 2(b) shows that the credit depletion period is not sufficient to correctly determine which resource must be throttled. The figure depicts CPU (left y-axis) and network (right y-axis) limitations as a function of the latency inflection point, essentially the credit depletion period. Limits previously analyzed (CPU utilization limits set to 30% and 40%, network bandwidth limit set to 200 Mbps) are highlighted for the sake of clarity. If only the latency inflection point is taken into account, one should limit the network bandwidth to 150 Mbps and observe performance degradation after 1500 seconds. However, such a limit makes the system latency too long to satisfy the defined SLO and the extended credit depletion period is useless. Optimal throttling should allow the user to decrease waste of resources while complying with the SLO.

Observation: Throttling allows delaying the credit depletion period but if done too agressively user SLOs are violated. Optimal throttling can be obtained by considering a huge search space that accounts for which resource to throttle, its throttling level, system workload, and system performance.

Fig. 2(a) shows that throttling different resources may affect the credit depletion period and system performance in unexpected ways. Although *cpulimit* = 40% (with network bandwidth = 192 Mbps) and Network Limitation = 200 Mbps (with CPU usage = 22%) have a similar 95th-percentile latency, the credit depletion period is 120 seconds longer when the network is throttled than when the CPU is throttled. The effect of each throttling on the system depends on the



(a) Ceph, *cpulimit* = 40% (b) Ceph, *Network Limitation* = 200 Mbps (c) TPC-W, *cpulimit* = 30%

Fig. 3: Available CPU and network credits as a function of time for Ceph and TPC-W.

application and system configuration (e.g., workload, system deployment, system hardware). Therefore, it is challenging to choose the most efficient throttling on a system in advance.

Observation: While the credit depletion period of a burstable instance depends strongly on the throttled resource, throttling different resources may have the same performance effect but different depletion periods.

Another parameter that determines the effectiveness of throttling and can decrease waste of resources is instance type. Fig. 2(c) depicts the 95th-percentile latency of Ceph when it is deployed on two different instances (i.e., *t2.small* and *t2.medium*) with different CPU throttling levels (i.e., *cpulimit* set to 30% and 50%). Here, two different SLOs are considered, namely SLO_1 and SLO_2. The small instance can comply with both SLOs depending on the CPU limitation. *Cpulimit* = 30% allows the system to satisfy SLO_1 for at least 15 minutes, but it does not comply with stricter SLOs. *Cpulimit* = 50% makes the small instance comply with both SLOs for 11 minutes. While this is not convenient for satisfying SLO_1 since the credit depletion period is now shorter, it is beneficial if the system must comply with SLO_2. Alternatively, the instance type may be upgraded from small to medium. Although a medium instance is more expensive than a small one, *t2.medium* always complies with SLO_2 when *cpulimit* = 30%.

Observation: Correct selection of instance type allows processing requests with stricter SLOs, this selection is not straight-forward when the monetary cost and credit depletion periods influence the choice.

Fig. 3 depicts the available network (left y-axis) and CPU (right y-axis) credits as a function of time for Ceph and TPC-W. It shows that different applications may require different resources to work properly. Ceph, see Figs. 3(a) and 3(b), always consumes network credits before CPU ones. Hence, the credit depletion period ends when network credits are no more available, i.e., between 800 and 920 seconds. TPC-W, see Fig. 3(c), mainly consumes CPU credits that are exhausted after 9600 seconds when *cpulimit* is set to 30%. A framework that monitors only network (CPU) credits can efficiently schedule Ceph (TPC-W) migration before performance deteriorates, but it is ineffective when TPC-W (Ceph) is executed.

Observation: Different applications use burstable resources in different ways. Monitoring credits of *all* resources is necessary to avoid SLO violations, especially when instance migration is considered.

To summarize the above: *i)* real applications need different resources to work correctly; *ii)* the optimal throttling can be determined only after considering a considerable search space; *iii)* it is nontrivial to determine the existent correlation among different resources and their effect on application performance; *iv)* credits are consumed at different rates by each resource. It is therefore critical to deploy a data analytics framework that can simultaneously tackle the following **challenges**:

- Which is the most efficient way to extend the instance credit depletion period (i.e., limiting system performance) while making the application comply with SLOs?
- How an enormous search space (i.e., workload, resource to be throttled, throttling level, instance type, latency) may be profiled in a short time to select optimal throttling and instance type?
- Which is the best instance type to satisfy SLOs and minimize cost?
- When should applications be migrated on a new instance to avoid SLO violations and optimize cost?

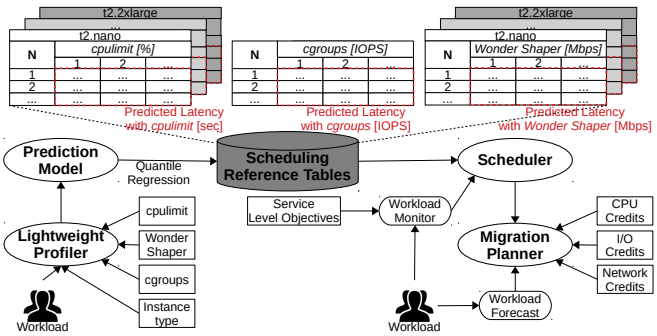


Fig. 4: Overview of CEDULE+.

IV. CEDULE+

CEDULE+ is a middleware framework that can be deployed on the burstable instance that hosts the application. The design of the framework allows nimbleness: it is based on lightweight profiling that can be performed offline and since it implements throttling with standard tools, it incurs a negligible overhead to the host instance. Given an application and its workload, CEDULE+ determines the optimal scheduling configuration (i.e., which instance type to use, which resource to throttle and with which throttling level) that maximizes the credit depletion period while minimizing the instance cost and migration penalty, as well as complying with the user-defined SLO.

Fig. 4 provides an overview of CEDULE+ and its module composition: the *Lightweight Profiler* profiles the system using a limited number of experiments; the *Prediction Model* uses the profiler output to generate Scheduling Reference Tables through multi-dimensional quantile regression; the *Scheduler* monitors the system workload and sets the optimal system configuration according to the Scheduling Reference Tables; the *Migration Planner* schedules the next migration based on available credits and predicted workload. CEDULE+ throttles CPU, network, and I/O usage dynamically according to the workload through *cpulimit*, *Wonder Shaper*, and *cgroups*. In the following, we elaborate on each of the four components.

A. Lightweight Profiler

Exhaustive profiling is expensive and time-consuming due to the enormous search space. If C cpulimit values, L workloads, and K instance types are considered, the size of the search space is $C \cdot L \cdot K$. In addition, a large number of samples need to be collected for each experiment in order to properly determine percentile latencies, especially for high percentile values. If t_C is the time required to achieve statistical stability in CPU profiling, then exhaustive profiling requires $C \cdot L \cdot K \cdot t_C$ time units to profile system performance when throttling CPU usage. One needs to repeat the same profiling process for W values of *Wonder Shaper* and D values of *cgroups*. When throttling I/O, the instance type does not affect system performance since I/O bandwidth depends only on the size of the volume attached to the instance. Therefore, the total time required to exhaustively profile the entire system is $(C \cdot K \cdot t_C + W \cdot K \cdot t_W + D \cdot t_D) \cdot L$, where t_W (t_D) is the time needed to achieve statistical stability when *Wonder Shaper* (*cgroups*) limits the network (I/O) bandwidth.

CEDULE+ adopts a lightweight profiling approach that collects *a few empirical measurements* of latency for sparse distinct values of *cpulimit*, *Wonder Shaper*, *cgroups*, and workload. Then, these measured latencies are fed into an analytic model to estimate the missing points. Assume the percentage of data collected for C, W, D, and L is σ, ω, δ, and λ, respectively. The lightweight strategy profiling time is $\lambda \cdot L \cdot (\sigma \cdot C \cdot K \cdot t_C + \omega \cdot W \cdot K \cdot t_W + \delta \cdot D \cdot t_D)$, i.e.,

$$\frac{\lambda \cdot (\sigma \cdot C \cdot K \cdot t_C + \omega \cdot W \cdot K \cdot t_W + \delta \cdot D \cdot t_D)}{C \cdot K \cdot t_C + W \cdot K \cdot t_W + D \cdot t_D}$$

times shorter than the exhaustive strategy. Note that re-profiling is required when the service process changes (e.g., application changes or AWS upgrades) as CEDULE+ captures the dynamics of the arrival workload.

B. Prediction Model

Table I gives a summary of the notation used by the prediction model. The credit depletion period of an instance (T^{dep}) is defined by the first resource *res* that exhausts all its credits: $T^{dep} = \min_{res}(T^{dep}_{res})$, with T^{dep}_{res} being the credit depletion period of a resource *res*. The distance between the credit depletion period of an instance and its migration time under a given SLO (T^{mig}) is called *Credit Efficiency*. The sum of the instance cost ($Cost_{vm} \cdot T^{dep}$) and the migration penalty ($Cost^{mig}$) due to the limited number of available migrations in AWS [14] is called *Cost*. In order to maximize the credit efficiency while minimizing the cost and complying with SLOs, we define the following linear optimization problem:

TABLE I: Notation and abbreviations.

Notation	Meaning
res	A burstable resource (i.e., CPU, network, or I/O)
T^{dep}_{res}	Credit depletion time for resource *res*
$T^{dep} = \min_{res}(T^{dep}_{res})$	Credit depletion time of the instance
$cr_{res}(t)$	Credits available at time t for resource *res*
R^{con}_{res}	Credit consumption rate for resource *res*
R^{gen}_{res}	Credit generation rate for resource *res*
$Util_{res}$	Utilization of resource *res*
BW_{res}	Bandwidth of resource *res*
T^{mig}	Migration time
$Cost_{vm}$	Cost of an instance (VM) per time unit
$Cost^{mig}$	Migration penalty
SLO	Service Level Objective
P^i	ith-percentile latency
$T^{dep} - T^{mig}$	Credit Efficiency
$Cost_{vm} \cdot T^{dep} + Cost^{mig}$	Cost

$$\begin{aligned} &\text{maximize} && T^{dep} - T^{mig} \\ &\text{subject to} && \text{minimize} \quad Cost_{vm} \cdot \mathrm{T}^{dep} + Cost^{mig} \\ &&& \text{subject to} \quad P^i \leq \text{SLO}, \end{aligned} \tag{1}$$

where P^i is the ith-percentile latency that depends on CPU usage, IOPS, and network bandwidth. The instance type hosting the application and resources throttling are the decision variables of the problem since they are controlled for optimizing the cost. T^{mig} is the instance migration time, which depends on the application type (i.e., stateless vs. stateful). Since CEDULE+ can predict the credit depletion time of an instance, it can proactively create a new virtual machine (VM) to migrate the application [27]. $T^{mig} = 0$ for stateless application since no memory is copied. The migration time of a stateful application is the time required to copy the application status.

Next, we show how to derive T^{dep}_{res} for all resources. T^{dep}_{CPU} can be derived through equation [53]:

$$T^{dep}_{CPU} = \frac{cr_{CPU}(t)}{R^{con}_{CPU} \cdot Util_{CPU} - R^{gen}_{CPU}}, \tag{2}$$

while $T^{dep}_{I/O}$ [47] and T^{dep}_{net} through equation:

$$T^{dep}_{res} = \frac{cr_{res}(t)}{R^{con}_{res} \cdot BW_{res} - R^{gen}_{res}}, \tag{3}$$

where $cr_{res}(t)$ is the number of available credits at time t for resource *res* (i.e., CPU, I/O, or network) and R^{con}_{res} (R^{gen}_{res}) is its credit consumption (generation) rate. $Util_{CPU}$ is the CPU usage and BW_{res} is the bandwidth of resource *res* (i.e., network or I/O). AWS provides $cr_{res}(0)$, R^{gen}_{res}, and R^{con}_{res} for CPU and I/O [47, 53], but the models and parameters for determining the network credit depletion are not disclosed. We derive the undisclosed parameters (i.e., $cr_{net}(0)$, R^{con}_{net}, and R^{gen}_{net}) using *iperf3* benchmark [56]. Specifically, we use iperf3 to send data from a client (which we deploy on a *m5.large* instance to avoid bursting performance) to a server placed on a burstable

TABLE II: Network credit parameters of AWS T2 instances.

	$cr_{net}(0) = cr_{net}^{max}$	R_{net}^{con}	R_{net}^{gen}
t2.nano	57776 cr	1 cr/Mbit	30.5 cr/sec
t2.micro	123617 cr	1 cr/Mbit	61 cr/sec
t2.small	115400 cr	1 cr/Mbit	122 cr/sec
t2.medium	98931 cr	1 cr/Mbit	244 cr/sec
t2.large	66027 cr	1 cr/Mbit	488 cr/sec
t2.xlarge	33032 cr	1 cr/Mbit	715 cr/sec
t2.2xlarge	N/A	1 cr/Mbit	976 cr/sec

instance. $cr_{net}(0)$ is obtained from Eq. (3) by setting $t = 0$ and multiplying both sides of the equation by $R_{net}^{con} \cdot BW_{net} - R_{net}^{gen}$:

$$cr_{net}(0) = T_{net}^{dep} \cdot \left(R_{net}^{con} \cdot BW_{net} - R_{net}^{gen}\right), \tag{4}$$

where T_{net}^{dep} is the time when we first observe performance degradation, BW_{net} is the average network bandwidth before credits are exhausted, R_{net}^{gen} is equal to the network bandwidth when no credits are available, and R_{net}^{con} is observed to be 1 $cr/Mbit$. Table II reports the network credit parameters for AWS T2 instances derived through Eq. (4).

We use quantile regression [2] to estimate the percentile latencies of unmeasured configurations during profiling. Quantile regression is a technique for statistical inference that is capable of inferring the correlation between conditional quantile functions. It avoids over-fitting, handles outliers in the training data (due to resource contention) better than linear regression, and does not need any assumptions on latency distribution [22] that is dynamic and unpredictable due to interference with other users [24].

Typically, quantile regression admits one-dimensional input data [30, 40]. For the use case of this paper, it is required to derive the limiting value for each throttled resource *and* instance type under various system loads. This requirement forms three two-dimensional spaces (i.e., one space per resource). CEDULE+ implements a multi-dimensional version of the classic quantile regression [2]: *i)* it assumes a steady load and uses sampled points to derive system performance when the throttling level varies; *ii)* the same procedure is repeated by fixing the throttling level and varying the system load; *iii)* the two models are combined to derive the global performance model of the considered resource. This methodology is applied to each resource and instance type to populate the *Scheduling Reference Tables* in Fig. 4.

C. Scheduler

The scheduler monitors the system load and chooses the optimal configuration that makes the system comply with the SLO while optimizing credit efficiency and cost. The scheduler selects *i)* the instance type that allows serving all the incoming requests and *ii)* the throttling parameters (i.e., throttling type and its level) that bring the ith-percentile latency as close as possible to the SLO (i.e., maximize the credit efficiency). When dynamic workloads are considered, the scheduler continuously monitors the workload to identify sudden variations and changes throttling configuration accordingly.

D. Migration Planner

CEDULE+ migrates instances in two cases: 1) when system performance deteriorates due to credit exhaustion and 2) to avoid SLO violations or waste of resources owing to workload variations. In the former case, CEDULE+ adopts Eqs. (2)–(3) to predict the credit depletion period of each resource. It schedules instance migration and launches a new instance in advance (e.g., at least 60 to 100 seconds based on the AWS virtual machines start-up time [7, 29]). This way, the migration time does not account for the instance start-up since CEDULE+ moves the application to the new VM only after its booting phase is completed.

If a stateless application needs to be migrated, CEDULE+ launches a new instance. Volumes storing persistent data are attached to the new instance. The migration is transparent to users since the incoming traffic is automatically redirected to the new instance before the old instance is turned off. Stateful applications need to migrate ephemeral data from the old instance. CEDULE+ accounts for resources (i.e., CPU, I/O, and network) required to migrate data. If the migration of a stateful application is triggered by the prediction of credit exhaustion, CEDULE+ hastens the migration such that it is completed before running out of resource credits, i.e., the credit depletion period ends. For example, a 1 MB file can be transferred in 4 ms if the maximum I/O throughput is 250 MB/s [46]. This operation takes 8 network credits (1 credit/Mbit, see Table II) and 64 I/O credits (1 credit/IOPS [47]). Provided that the CPU usage is negligible (i.e., no CPU credits are consumed), CEDULE+ starts the data migration when there are enough network and I/O credits for completing the operation without violating the SLO, i.e., as soon as there are only 8 network credits or 64 I/O credits.

E. Vertical Scaling vs. Horizontal Scaling

When workload variations are observed, CEDULE+ can opt for vertical or horizontal scaling (i.e., scale-up and scale-out, respectively). Small workload variations can be addressed quickly with vertical scaling, while horizontal scaling allows managing unexpected workload peaks [42]. When CEDULE+ scales-up and migrates the application to a more powerful instance, it follows the same steps described for migrations due to credit exhaustion. If scale-out is preferred, i.e, the same application is replicated to an additional instance, CEDULE+ allows the new instance to access persistent data by copying them to a new volume before turning on the new instance. Although AWS provides Multi-Attach volumes [50] (i.e., volumes that can be attached to multiple instances), T2 instances (considered in this paper) are not supported. CEDULE+ uses vertical scaling for all experiments presented in Section V.

V. Experimental Evaluation

The efficiency of CEDULE+ is assessed through experiments on Amazon EC2. We assess the accuracy of CPU and network credit depletion models given in Eqs. (2) and (3), respectively, and evaluate the error of the prediction methodology outlined in Section IV. The performance of CEDULE+ is analyzed using both static and dynamic workloads, and its efficiency is compared to the default AWS strategy. Although CEDULE+ is deployed only atop Amazon EC2, it can manage burstable instances of any other provider (e.g., Azure [62] or

TABLE III: T2 instances characteristics [48]. The baseline performance refers to each available vCPU.

T2 type	vCPU	Init. Cr.	R^{gen}_{CPU} [cr/hr]	Baseln. Perf.	Max Cr.
nano	1	30	3	5%	72
micro	1	30	6	10%	144
small	1	30	12	20%	288
medium	2	60	24	20%	576
large	2	60	36	30%	864
xlarge	4	120	54	22.5%	1296
2xlarge	8	240	81.6	17%	1958.4

GCE [60]). The functioning of CEDULE+ stays unchanged, but its scheduling reference tables must be trained by profiling the desired cloud environment. Parameters that are not disclosed by other providers, e.g., $cr_{net}(0)$, R^{con}_{net}, or R^{gen}_{net}, must be derived.

A. Experimental Setup

The operating system of each Amazon T2 instance is Ubuntu Server 18.04 LTS. All VMs are placed in the *us-east-1* region and share the same subnet. Network performance of T2 instances is obtained through experiments and shown in Table II. AWS [53] provides details about the CPU capacity of T2 instances (see Table III). The max I/O bandwidth depends on the volume size [47] and is: $\max(3 \cdot \textit{volume size}, 100)$ IOPS.

1) Applications: We use three applications: TPC-W [1], FTP, and Ceph [4] to evaluate CEDULE+.

TPC-W is a web benchmark that models a three-tier e-commerce application, i.e., client generating requests, front-end web servers, and database servers. We instantiate one web server and one database on two separate burstable instances, and use the *ordering* workload profile with a read:update request ratio of 1:1. Since the CPU utilization of client is always below 10%, there is no need to throttle the CPU usage of this tier. To consider a balanced system, the same *cpulimit* value is applied on web and database servers. *Wonder Shaper* does not affect TPC-W performance since the bandwidth needed by this application is smaller than the instance baseline.

FTP is based on the client/server paradigm. An *FTP daemon* is installed on the server and listens for requests from remote clients. We use FTP to generate downloading requests to get a file (10 MB) from the FTP server. Page cache is disabled before running the FTP daemon to consume both network and I/O credits (i.e., bypass cache and access I/O). The FTP server is deployed on a burstable instance, while the client is on a *m5.large* to avoid performance interference between client and server. The number of simultaneous downloading requests (users) can be adjusted on the client side to change the workload intensity. A 40 GB gp2 volume (i.e., 120 IOPS baseline) is attached to the FTP server. Since FTP hardly uses any CPU cycles, CPU usage is not throttled. *Wonder Shaper* and *cgroups* are used to limit network and I/O.

Ceph is a platform that implements distributed object storage with high efficiency and reliability. Clients communicate directly with Object Storage Devices (OSDs) to read and write information while a Metadata Server (MDS) handles metadata operations (e.g., open and rename). CPU and I/O execute client read and write requests, while the network enables communication between clients and OSDs. The cluster used for these experiments is composed by a Monitor/Client installed on a *m5.large* instance and an OSD deployed on a burstable VM. Ceph Mimic (i.e., v13.2.4) is the Ceph version installed on each cluster node. The cluster stores information in one pool and does not replicate objects as it consists of only one OSD. The OSD mounts a 100 GB gp2 volume (i.e., 300 IOPS baseline). RADOS [5] is a benchmark that is specifically intended to test Ceph clusters and it is used here to assess Ceph performance. RADOS defines the type of operation, the number of simultaneous users of the system, the size of each object (read or written by the user), and the benchmark duration. While Ceph consumes credits of all resources (i.e., CPU, I/O, and network), it is nontrivial to throttle I/O performance using *cgroups* as I/O usage depends on the OSD file system and its storage back-end (i.e., BlueStore [64]). For this reason, we only evaluate how CPU and network affect performance. This does not affect the evaluation of CEDULE+ since the volume size of the OSD is sufficiently large to make the I/O credit depletion period longer than those of network and CPU.

2) SLO: There are not standard SLOs for any of the three benchmarks described in Section V-A1. For this reason, we always set the SLO to a few seconds. CEDULE+ can work with *any* user-defined SLO.

3) Workload: We evaluate CEDULE+ with static and dynamic loads for TPC-W, FTP, and Ceph. All applications have as configuration parameter the number of concurrent users which defines the workload intensity.

Static load: The number of concurrent users is fixed for the duration of the experiments. These experiments allow training the quantile regression model and evaluate prediction accuracy.

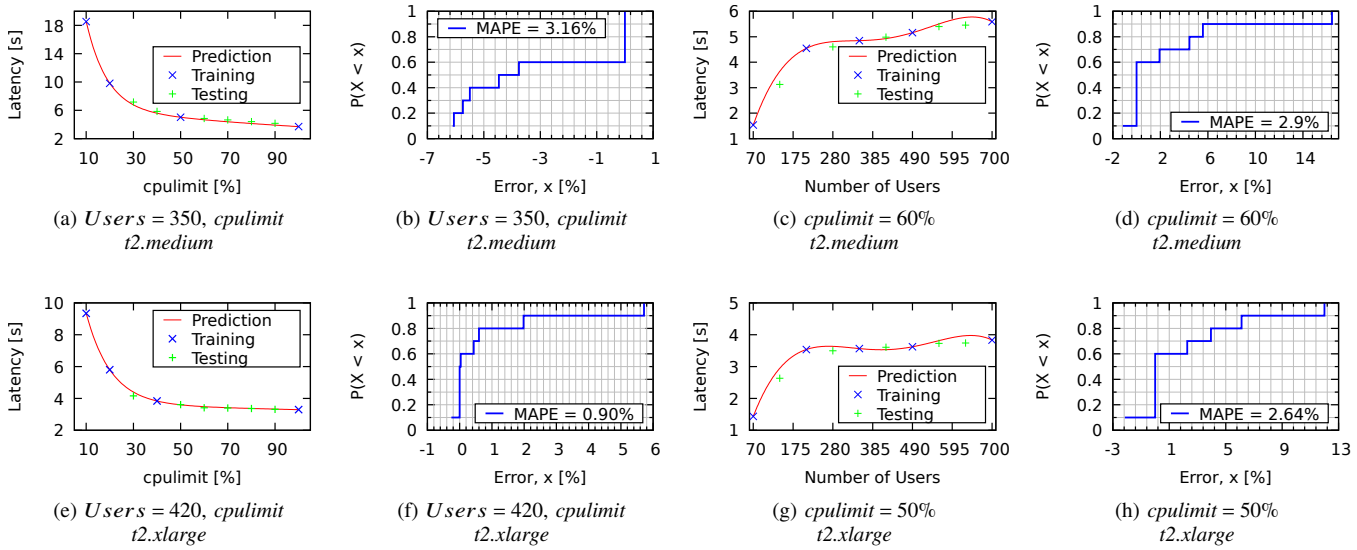
Dynamic load: Real-world applications serve dynamic workloads. The number of users varies across time to evaluate CEDULE+ performance under varying workload intensities.

TABLE IV: Accuracy of CPU and network credit depletion period model for different applications and limiting strategies.

res	App.	Thr. Res. (Thr. Lvl.)	$T^{dep,estim}_{res}$	$T^{dep,real}_{res}$	Error
CPU	TPC-W	CPU (100%)	1680 s	1620 s	3.6%
CPU	TPC-W	CPU (62%)	2880 s	2700 s	6.3%
net	FTP	net (250 Mbps)	757 s	765 s	1.0%
net	FTP	I/O (300 IOPS)	497 s	521 s	4.4%
net	Ceph	net (400 Mbps)	403 s	401 s	0.5%
net	Ceph	CPU (30%)	286 s	296 s	3.4%

B. Accuracy of Credit Depletion Period Models

We evaluate the accuracy of credit depletion period models for CPU and network, see Eqs. (2) and (3), when resources of the three applications are throttled. The estimated credit depletion period is compared with the observed one via experimentation. Results are shown in Table IV for different resources and throttling levels. The error, defined as $|T^{dep,estim}_{res} - T^{dep,real}_{res}|/T^{dep,real}_{res}$, is never larger than 10% and corroborates the accuracy of CPU and network models. Since AWS publishes the I/O credit depletion model [47], its accuracy is not evaluated here.



(a) $Users = 350$, *cpulimit* *t2.medium*
(b) $Users = 350$, *cpulimit* *t2.medium*
(c) *cpulimit* = 60% *t2.medium*
(d) *cpulimit* = 60% *t2.medium*
(e) $Users = 420$, *cpulimit* *t2.xlarge*
(f) $Users = 420$, *cpulimit* *t2.xlarge*
(g) *cpulimit* = 50% *t2.xlarge*
(h) *cpulimit* = 50% *t2.xlarge*

Fig. 5: CEDULE+ prediction of the 95th-percentile latency of TPC-W with static workload. The mean absolute error is reported.

C. Static Workload: Accuracy of Quantile Regression

Quantile regression is in the heart of CEDULE+ as it is used to populate the Scheduling Reference Table, see Fig. 4. Here, we evaluate the effectiveness of quantile regression for different applications, throttled resources, throttling levels, and instance types. The error of prediction is computed as: $err(l) = (Y(l) - y_l)/y_l$, where $Y(l)$ and y_l are the predicted and measured 95th-percentile latencies for the throttling parameter l. Negative error values indicate the regression model underestimates the 95th-percentile latency, while positive values denote over-estimation. The mean absolute percentage error (MAPE) is also computed by averaging $|err(l)|$.

Fig. 5 shows the accuracy of quantile regression when CEDULE+ monitors TPC-W. TPC-W is a CPU intensive workload, it is therefore reasonable to limit its CPU utilization to increase credit efficiency. The prediction model is trained for *cpulimit* = $\{10, 20, 50, 100\}$% and $N = \{70, 210, 350, 490, 700\}$ users. Each experiment lasts 600 seconds. The first and second columns in Fig. 5 show the accuracy of the model for different instance types (i.e., *t2.medium* and *t2.xlarge*), when the number of users is fixed and the latency varies as a function of *cpulimit*. Similarly, the third and fourth columns in Fig. 5 present the accuracy of the model when *cpulimit* is fixed and the load varies. In all cases, the MAPE is not larger than 4%.

Fig. 6 depicts results for Ceph. For this workload, credit efficiency can be improved by throttling either CPU or network, since this application needs both resources to process requests. The prediction model is trained for *cpulimit* = $\{10, 15, 35, 55, 100\}$% or *Wonder Shaper* = $\{61, 100, 200, 500\}$ Mbps, and $N = \{10, 20, 50, 80, 100, 300, 1600, 12800\}$. Each experiment lasts 200 seconds. Fig. 6 shows the regression model accuracy for different instance capacities (i.e., *t2.small*, *t2.medium*, and *t2.large*), CPU usage, network bandwidth, and workloads. In all cases, the model predicts system performance efficiently and its mean absolute error is smaller than 10%. The prediction model works better when CEDULE+ uses *Wonder Shaper* to throttle system performance.

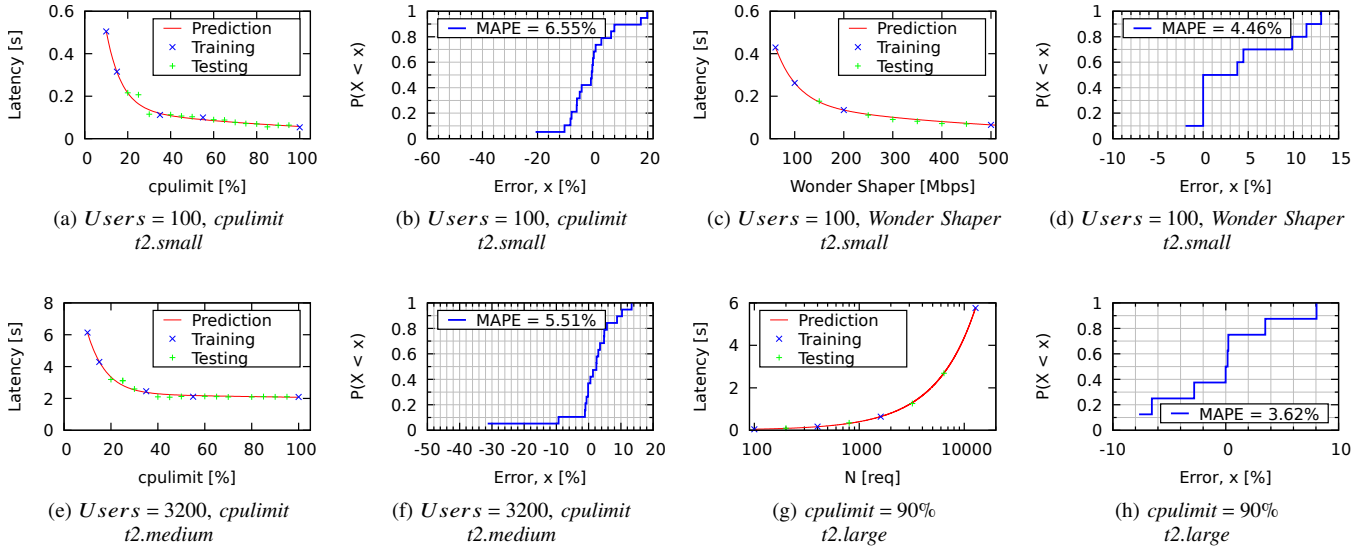
The prediction error is smaller than 5% for FTP (results are not reported due to lack of space). In this case, the credit efficiency can be increased by throttling network and I/O bandwidths using *Wonder Shaper* and *cgroups*.

D. CEDULE+ with Dynamic Workload

After creating the Scheduling Reference Tables as described in Section IV, CEDULE+ can handle dynamic workloads where the number of users that simultaneously connect to the system fluctuates. To highlight the efficiency of CEDULE+, we initially assume that only one type of instance (i.e., *t2.micro*) is used for the application deployment, then we allow CEDULE+ to select the best instance type.

Fig. 7 shows the 95th-percentile of FTP when only *t2.micro* instances are used for its deployment. The 95th-percentile latency is normalized over the SLO and is plotted against the left y-axis. The system workload is plotted against the right y-axis. It is generated by varying the number of users (from 10 to 100) who are simultaneously downloading a file. The time duration of workload variation is selected randomly between 180 and 350 seconds. CEDULE+ controls FTP latency by limiting the resource (I/O or network) that provides the greatest credit efficiency with the actual workload. The effect of the two different limitations is shown in Fig. 7. The shaded region between the two latency lines reflects the number of resources saved by each strategy. If the region is blue, throttling network bandwidth provides higher credit efficiency. Otherwise, it is preferable to limit I/O. Note that the application complies with the given SLO independently of the throttled resource.

Fig. 8 depicts the system workload (right y-axis) and the 95th-percentile latency of TPC-W and Ceph (left y-axis) when CEDULE+ can deploy them on any type of instance. Fig. 8(a) shows the effect of CEDULE+ on TPC-W latency. The number of concurrent requests varies randomly between 70 and 510 every 3 to 10 minutes. Now, CEDULE+ can also select the



(a) $Users = 100$, *cpulimit* *t2.small*
(b) $Users = 100$, *cpulimit* *t2.small*
(c) $Users = 100$, *Wonder Shaper* *t2.small*
(d) $Users = 100$, *Wonder Shaper* *t2.small*
(e) $Users = 3200$, *cpulimit* *t2.medium*
(f) $Users = 3200$, *cpulimit* *t2.medium*
(g) *cpulimit* = 90% *t2.large*
(h) *cpulimit* = 90% *t2.large*

Fig. 6: CEDULE+ prediction of the 95th-percentile latency of Ceph with static workload. The mean absolute error is reported.

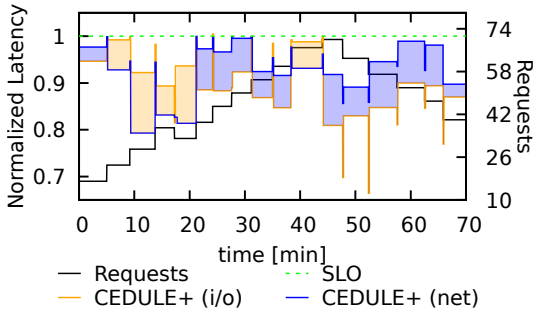


Fig. 7: 95th-percentile latency (left y-axis) of FTP with dynamic workload (simultaneous user requests, right y-axis). Latency is normalized over the SLO (simple ratio). FTP is deployed on *t2.micro*. Instance migration is shown in Fig. 9(b).

instance type that more efficiently process the system load. As the workload changes, CEDULE+ migrates the application on a larger (smaller) instance to comply with the SLO while optimizing costs. The letters atop the graph indicate the type of instance used to deploy the application. *S* is for *t2.small*, *M* is for *t2.medium*, and *xL* is for *t2.xlarge*. Dashed vertical lines show the time when the instance type changes. CEDULE+ keeps TPC-W latency close to the SLO (without violating it) while using the most inexpensive CPU resources.

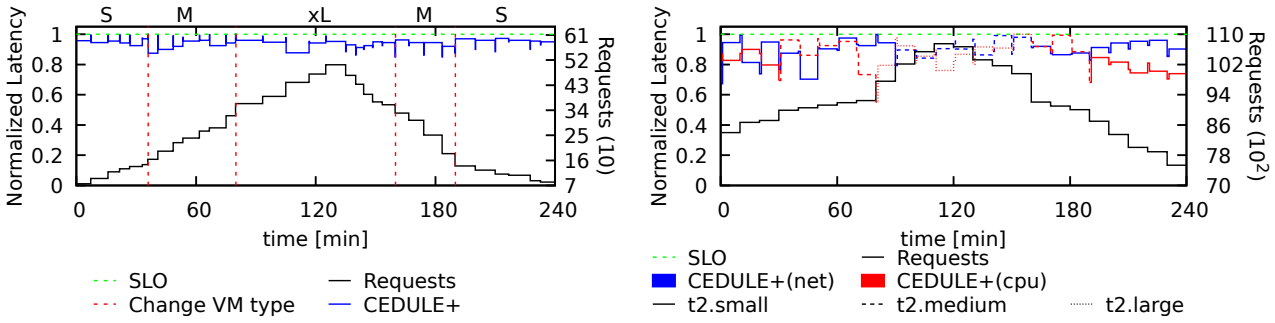
Fig. 8(b) shows latency for Ceph when CEDULE+ controls the system by limiting network (blue line) and CPU (red line). The credit depletion period is extended by limiting either resource, but throttling network bandwidth allows CEDULE+ to reach better performance. Generally, network throttling makes the system latency closer to the SLO and allows meeting the objective with smaller instances. CEDULE+ always selects the cheapest instance type that allows complying with the SLO and throttles the resource that increases credit efficiency the most. For example, from 0 to 30 minutes, the SLO may be satisfied deploying Ceph on *t2.small* independently of the throttled resource. In this case, CEDULE+ limits either CPU or network after evaluating which limitation provides the largest credit efficiency. From 30 to 60 minutes, throttling the CPU allows decreasing the number of wasted resources more than throttling the network, i.e., the system latency is closer to the SLO when CPU usage is limited. Note that the system can be deployed on instances with smaller capacity if CEDULE+ limits the network bandwidth. For this reason, CEDULE+ throttles the network and chooses to reduce costs instead of further increasing credit efficiency.

In these experiments, applications are deployed on shared resources (i.e., there is no dedicated hardware). Although different strategies exist to increase the performance isolation of multi-tenant clouds [12, 10, 17, 15], CEDULE+ remains robust even when performance interference is observed.

E. Lightweight Profiling vs. Exhaustive Profiling

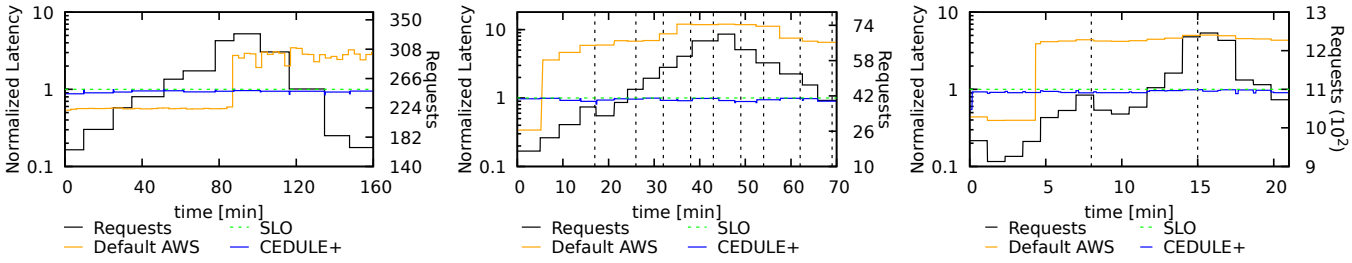
Lightweight profiling allows training CEDULE+ with a much lower cost than exhaustive profiling. For example, when profiling Ceph, CEDULE+ profiles 5 distinct CPU throttling levels, 4 network levels, and 8 different workloads (see Section V-C). That is, 40 profiling experiments for CPU and 32 for network for each instance type (i.e., 7). Since each experiment lasts 200 seconds, profiling Ceph requires 28 hours.

When exhaustive profiling is adopted, the time required to complete the application profiling is 800 times longer since each combination of limit and load values must be profiled. For the sake of simplicity, assume that the network bandwidth cannot be larger than 500 Mbps and that the maximum number of users connected to Ceph is 100. In this case, 70K experiments must be run for *cpulimit* (i.e., 100 CPU usage values, 100 different loads, and 7 instance types), and 350K experiments are required when *Wonder Shaper* is applied. This exhaustive profiling is completed after 2.66 years.



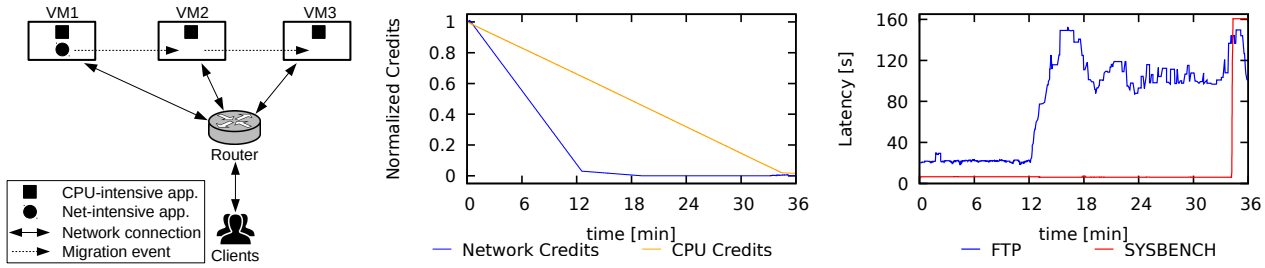
(a) TPC-W (b) Ceph

Fig. 8: 95th-percentile latency (left y-axis) of (a) TPC-W and (b) Ceph with dynamic workload (simultaneous user requests, right y-axis). Latency is normalized over the SLO (simple ratio). CEDULE+ needs 1 minute to adjust the limiting value to the new workload. Letters atop (a) describe the instance type used to deploy TPC-W in each period delimited by red dashed lines: S is *t2.small*, M is *t2.medium*, and xL is *t2.xlarge*. In (b), the line dashing style represents the instance type, see the legend.



(a) TPC-W (b) FTP (c) Ceph

Fig. 9: 95th-percentile latency obtained with different strategies for various applications (left y-axis in logarithmic scale) with fluctuating workload (right y-axis). Results are normalized over the SLO (simple ratio) and the workload is the number of users connecting to the system concurrently. Vertical dashed lines show when CEDULE+ migrates the VM to a new instance.



(a) Case-study (b) Credit depletion period (c) 95th-percentile latency

Fig. 10: CEDULE+ with co-located heterogeneous applications. The latency is normalized over the SLO (simple ratio).

F. Improvements over the Default AWS Policy

CEDULE+ is compared to the default Amazon strategy. Fig. 9 depicts the workload (right y-axis) and 95th-percentile latency normalized over the SLO (left y-axis) of each application. Fig. 9(a) shows TPC-W results. Initially, the system satisfies the user-defined SLO independently of the adopted strategy. Amazon's default strategy depletes the CPU credits faster than CEDULE+ and violates the SLO after 83 minutes. The default strategy does not monitor the available CPU credits and cannot migrate the application (i.e., users must do that manually). CEDULE+ always complies with the SLO, extends the credit depletion period of the instance by almost 2 times, and migrates the application after 160 minutes.

Similar results are observed for FTP and Ceph, see Figs. 9(b) and 9(c), respectively. In both cases, performance degrades after a few minutes due to network credits exhaustion when CEDULE+ is not used. CEDULE+ instead optimizes resource usage and migrates applications when required (the migration time is indicated by vertical dashed lines). The credit depletion period of instances that use CEDULE+ is 2.4 and 0.95 times longer for FTP and Ceph, respectively, than the period observed with the default Amazon strategy.

CEDULE+ also allows decreasing the monetary cost when different instance types are used for application deployment. Consider the TPC-W application in Fig. 9(a). When a strategy that cannot change the instance type is adopted, one must deploy the application on an instance that allows complying with the SLO in the worst-case scenario (i.e., 510 concurrent users into the system). TPC-W should be deployed on a *t2.xlarge*. The experiment runs for 240 minutes (4 hours) on instances in the us-east-1 region with Linux OS, and the cost is [48]: 4 hr $\cdot$ 0.1856 \$/hr = \$0.74. If CEDULE+ manages the system, the application runs 80 minutes on each instance (i.e., *t2.small*, *t2.medium*, and *t2.xlarge*). The cost is 54% cheaper:

$$1.33 \text{ hr} \cdot (0.023 \text{ \$/hr} + 0.0464 \text{ \$/hr} + 0.1856 \text{ \$/hr}) = \$0.34.$$

CEDULE+ may also benefit cloud service as a longer credit depletion period and new instances launched less frequently facilitate the management of spare resources that are used for burstable performance.

G. Heterogeneous Applications

Another way to increase credit efficiency is by co-locating heterogeneous jobs on the same VM, e.g., co-locate a CPU-intensive application with a network-intensive one. Since CEDULE+ enables throttling and monitoring of credit consumption, it increases the credit efficiency of each resource by migrating the application only when its credits are exhausted.

Here, we consider the system in Fig. 10(a), that is composed of three identical instances (VM1, VM2, and VM3), each hosting *sysbench* [57], a CPU-intensive benchmark. An FTP server is initially deployed on VM1 and migrated when the network credits of the hosting instance are exhausted. Fig. 10(b) depicts the CPU and network credit depletion time of VM1, i.e., the instance that initially hosts both applications. The FTP server consumes the network credits with a rate that is almost three times faster than the one required by *sysbench* to exhaust CPU credits. After 12 minutes, FTP performance deteriorates due to the unavailability of network credits, see Fig. 10(c), and the FTP server is migrated to VM2 to comply with the desired SLO. As expected, the performance of *sysbench* is not affected. Similarly, when FTP consumes all VM2 network credits the server is moved to VM3.

CEDULE+ and co-location of heterogeneous applications enable a twofold benefit for end-users. First, depending on the number of allocated instances, cost can be reduced. Here, cost is reduced by 25% since the user pays only for three instances (i.e., VM1, VM2, and VM3) instead of four (i.e., one for each application). Second, the reduced number of new instance initialization is an additional benefit. In this case-study, CEDULE+ allows the end-user to decrease the number of instance initialization by 50%. This is beneficial since AWS limits the number of new VMs that a user can launch (with free initial credits) during a 24-hour period [14].

VI. Related Work

AWS started offering burstable instance types in 2010 with *t1.micro* instances. Since 2014, all main cloud providers [49, 54, 59] offer similar instance types. Wen et al. [16] examine T1 instances (i.e., the AWS first-generation burstable instances) and propose to delay request execution to reduce costs. Jiang et al. [41] analyze CPU utilization and available CPU credits of T2 instances (i.e., the AWS second-generation burstable instances), and propose a model to study burstable instance performance, characterize instance type selection, and maximize the provider revenue. Mathá et al. [44] design a simulation environment for analyzing the performance of burstable instances. Baarzi et al. [38] develop BurScale, a framework that uses burstable instances to process requests when there is a substantial queue.

Several works focus on improving T2 instance credit management to increase VM performance. Leitner and Scheuner [13] propose a model for T2 instances and examine boosting performance by restarting VMs when all credits are consumed. Such a strategy is not useful anymore since AWS has implemented constraints on rebooting T2 instances [14]. The lifetime of CPU credits is stretched out in [27] using *cpulimit* to control CPU usage. *Cpulimit* outperforms the delay strategy proposed in [16]. Wang et al. [26] call attention on the mechanism controlling the credit depletion of VMs, showing that it follows a token-bucket model and in [25] they propose passive backup for spot instances to use burstable instances. This is the only work that studies the impact of burstable network on system performance but there is no focus on diminishing the network credit consumption rate.

Resource management of VMs in cloud computing has been researched thoroughly in the literature and many distinct approaches have been proposed [6, 8, 9, 11, 18, 34]. Javadi et al. [24] implement DIAL, an interference-aware load balancer to reduce long tail latencies. Wang et al. [21] make resource management more efficient by grouping VMs according to resource sharing and collocation criteria. Morris et. al. [19, 31] develop sprinting strategies using DVFS to boost processor clock rates and AWS burstable VMs. Assuming a priori knowledge of system characteristics (e.g., arrival rate), they perform off-line measurements and use machine learning to predict latencies. Dynamic capacity is investigated in [33] as a control knob, together with dynamic price, for maximizing profits of cloud providers. Tadakamalla and Menascé [32] design two controllers able to estimate requirements of a system subject to workload variations. This approach accounts only for the workload and does not consider other dimensions that affect the performance and monetary cost of burstable VMs. Ambati et al. [43] consider the expected workload to select the instance purchasing option (e.g., on-demand, reserved, spot) that allows optimizing long-term cloud costs, but do not consider burstable instances. The performance and cost of executing batch and interactive jobs on transient servers are studied in [36, 37]. Auto-scaling frameworks that minimize the cost of machine learning applications while meeting user-defined SLOs are proposed in [20, 23, 39]. These frameworks do not allow deploying applications on burstable instances.

Previous work considers migration to continuously provide service even when the selected VM is no longer available. Ray et al. [45] develop a proactive fault tolerance system by considering instance migration and its cost. Here, we propose autonomous migration to restore the performance of the in-

stance and optimize costs. Since CEDULE+ increases system efficiency, the total number of migrations is reduced compared to the case where users engage in migration manually.

To the best of our knowledge, there are no available frameworks that are SLO-aware and select the optimal burstable instance type based on the workload while throttling system performance to increase its credit efficiency. Our previous works CEDULE [28] and MRburst [35] perform preliminary study on applying CPU, I/O, and network throttling to increase the credit depletion time while meeting user-defined SLOs. However, these preliminary studies only support deploying applications to *t2.micro* instances, which results in limited application scalability (e.g., workload surges cannot be handled through instance upgrades) and increased resource wasting (e.g., the monetary cost cannot be reduced by downgrading the instance). CEDULE+ is a framework based on a data-driven analytics that combines multi-resource performance throttling, multi-instance migration, and quantile regression to enable the system to save credits while meeting user-defined SLOs.

VII. Concluding Remarks

This paper presents CEDULE+, a data analytics framework that extends the performance of burstable instances in cloud environments by throttling resource utilization. CEDULE+ can choose which burstable instance must be used for deploying the application, which resource (i.e., CPU, network, or I/O) must be throttled, and which throttling level should be applied. CEDULE+ adopts lightweight profiling and quantile regression to forecast application latency to selects the optimal combination of parameters to extend the credit depletion period of burstable instances and comply with the user-defined SLO. It also monitors the credit usage to efficiently schedule instance migration to minimize downtime.

We show the efficiency of CEDULE+ on AWS with three applications: TPC-W, FTP, and Ceph. In all cases, the prediction model is consistently accurate, with errors never larger than 10%. In the experiment scenarios presented in this paper, CEDULE+ extends the credit depletion period of an instance up to 240% and reduces deployment cost by more than 50%.

Acknowledgement

This work is supported in part by the following grants: National Science Foundation IIS-1838024 (using resources provided by Amazon Web Services as part of the NSF BIGDATA program), IIS-1838022, CCF-1756013, CCF-1717532, CNS-1950485, and the MIUR PRIN project SEDUCE 2017TWRCNB. We thank the anonymous reviewers for their insightful comments and suggestions.

References

[1] Wayne D Smith. *TPC-W: Benchmarking an ecommerce solution*. 2000.

[2] Roger Koenker and Kevin F Hallock. “Quantile regression”. In: *Journal of economic perspectives* 15.4 (2001), pp. 143–156.

[3] Marius A Eriksen. “Trickle: A Userland Bandwidth Shaper for UNIX-like Systems.” In: *USENIX Annual Technical Conference, FREENIX Track*. 2005, pp. 61–70.

[4] Sage Weil et al. “Ceph: A scalable, high-performance distributed file system”. In: *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association. 2006, pp. 307–320.

[5] Sage Weil et al. “Rados: a scalable, reliable storage service for petabyte-scale storage clusters”. In: *Proceedings of the 2nd International Workshop on Petascale Data Storage*. ACM. 2007, pp. 35–44.

[6] Robert Birke, Lydia Y. Chen, and Evgenia Smirni. “Data Centers in the Cloud: A Large Scale Performance Study”. In: *2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, June 24-29, 2012*. IEEE Computer Society, 2012, pp. 336–343.

[7] Ming Mao and Marty Humphrey. “A performance study on the vm startup time in the cloud”. In: *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE. 2012, pp. 423–430.

[8] Robert Birke et al. “State-of-the-practice in data center virtualization: Toward a better understanding of VM usage”. In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, June 24-27, 2013*. IEEE Computer Society, 2013, pp. 1–12.

[9] Robert Birke et al. “(Big)data in a virtualized world: volume, velocity, and variety in cloud datacenters”. In: *Proceedings of the 12th USENIX conference on File and Storage Technologies, FAST 2014, Santa Clara, CA, USA, February 17-20, 2014*. USENIX, 2014, pp. 177–189.

[10] Jacob Leverich and Christos Kozyrakis. “Reconciling high server utilization and sub-millisecond quality-of-service”. In: *Proceedings of the Ninth European Conference on Computer Systems*. 2014, pp. 1–14.

[11] Lei Lu et al. “Application-driven dynamic vertical scaling of virtual machines in resource pools”. In: *2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014*. IEEE, 2014, pp. 1–9.

[12] Miguel Gomes Xavier, Marcelo Veiga Neves, and Cesar Augusto Fonticielha De Rose. “A performance comparison of container-based virtualization systems for mapreduce clusters”. In: *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2014, pp. 299–306.

[13] Philipp Leitner and Joel Scheuner. “Bursting with Possibilities–An Empirical Study of Credit-Based Bursting Cloud Instance Types”. In: *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE. 2015, pp. 227–236.

[14] Rohit K Mehta and John Chandy. “Leveraging checkpoint/restore to optimize utilization of cloud compute resources”. In: *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*. IEEE. 2015, pp. 714–721.

[15] Abhishek Verma et al. “Large-scale cluster management at Google with Borg”. In: *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015*. ACM, 2015, 18:1–18:17.

[16] Jiawei Wen et al. “Less can be more: Micro-managing vms in amazon ec2”. In: *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*. IEEE. 2015, pp. 317–324.

[17] Miguel G Xavier et al. “A performance isolation analysis of disk-intensive workloads on container-based clouds”. In: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE. 2015, pp. 253–260.

[18] Ji Xue et al. “PRACTISE: Robust prediction of data center time series”. In: *11th International Conference on Network and Service Management, CNSM 2015, Barcelona, Spain, November 9-13, 2015*. IEEE Computer Society, 2015, pp. 126–134.

[19] Nathaniel Morris et al. “Sprint ability: How well does your software exploit bursts in processing capacity?” In: *2016 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2016, pp. 173–178.

[20] Shivaram Venkataraman et al. “Ernest: Efficient performance prediction for large-scale advanced analytics”. In: *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 2016, pp. 363–378.

[21] Yi Wang et al. “Demonstrating Scalability and Efficiency of Pack-Centric Resource Management for Cloud”. In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE. 2016, pp. 849–854.

[22] Yunqi Zhang et al. “Treadmill: Attributing the source of tail latency through precise load testing and statistical inference”. In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2016, pp. 456–468.

[23] Arpan Gujarati et al. “Swayam: distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency”. In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. ACM. 2017, pp. 109–120.

[24] Seyyed Ahmad Javadi and Anshul Gandhi. “DIAL: Reducing Tail Latencies for Cloud Applications via Dynamic Interference-aware Load Balancing”. In: *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2017, pp. 135–144.

[25] Cheng Wang et al. “Exploiting Spot and Burstable Instances for Improving the Cost-efficacy of In-Memory Caches on the Public Cloud”. In: *Proceedings of the Twelfth European Conference on Computer Systems*. ACM. 2017, pp. 620–634.

[26] Cheng Wang et al. “Using Burstable Instances in the Public Cloud: Why, When and How?” In: *POMACS* 1.1 (2017), 11:1–11:28.

[27] Feng Yan et al. “How to Supercharge the Amazon T2: Observations and Suggestions”. In: *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE. 2017, pp. 278–285.

[28] Ahsan Ali et al. “Cedule: A scheduling framework for burstable performance in cloud computing”. In: *2018 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2018, pp. 141–150.

[29] Andrew Chung, Jun Woo Park, and Gregory R Ganger. “Stratus: cost-aware container scheduling in the public cloud”. In: *Proceedings of the ACM Symposium on Cloud Computing*. ACM. 2018, pp. 121–134.

[30] Jean-Emile Dartois et al. “Using quantile regression for reclaiming unused cloud resources while achieving sla”. In: *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2018, pp. 89–98.

[31] Nathaniel Morris et al. “Model-driven computational sprinting”. In: *Proceedings of the Thirteenth EuroSys Conference*. ACM. 2018, 38:1–38:13.

[32] Venkat Tadakamalla and Daniel A Menascé. “Model-driven elasticity control for multi-server queues under traffic surges in cloud environments”. In: *2018 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE. 2018, pp. 157–162.

[33] Cheng Wang et al. “Effective Capacity Modulation as an Explicit Control Knob for Public Cloud Profitability”. In: *ACM Trans. Auton. Adapt. Syst.* 13.1 (2018), 2:1–2:25.

[34] Ji Xue et al. “Spatial-Temporal Prediction Models for Active Ticket Managing in Data Centers”. In: *IEEE Trans. Netw. Serv. Manag.* 15.1 (2018), pp. 39–52.

[35] Ahsan Ali et al. “It’s not a Sprint, it’s a marathon: Stretching Multi-resource Burstable Performance in Public Clouds”. In: *Proceedings of the 20th International Middleware Conference*. ACM. 2019.

[36] Pradeep Ambati and David Irwin. “Optimizing the Cost of Executing Mixed Interactive and Batch Workloads on Transient VMs”. In: *POMACS* 3.2 (2019), 28:1–28:24.

[37] Pradeep Ambati et al. “Understanding Synchronization Costs for Distributed ML on Transient Cloud Resources”. In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2019, pp. 145–155.

[38] Ataollah Fatahi Baarzi, Timothy Zhu, and Bhuvan Urgaonkar. “BurScale: Using Burstable Instances for Cost-Effective Autoscaling in the Public Cloud”. In: *Proceedings of the ACM Symposium on Cloud Computing*. 2019, pp. 126–138.

[39] Anirban Bhattacharjee et al. “BARISTA: Efficient and Scalable Serverless Serving System for Deep Learning Prediction Services”. In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2019, pp. 23–33.

[40] Jean-Emile Dartois et al. “Cuckoo: Opportunistic mapreduce on ephemeral and heterogeneous cloud resources”. In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE. 2019, pp. 396–403.

[41] Yuxuan Jiang et al. “Burstable instances for clouds: Performance modeling, equilibrium analysis, and revenue maximization”. In: *2019 IEEE Conference on Computer Communications, INFOCOM 2019*. IEEE. 2019, pp. 1576–1584.

[42] Fabiana Rossi, Matteo Nardelli, and Valeria Cardellini. “Horizontal and vertical scaling of container-based applications using reinforcement learning”. In: *2019 IEEE*

12th International Conference on Cloud Computing (CLOUD). IEEE. 2019, pp. 329–338.

[43] Pradeep Ambati et al. “Hedge Your Bets: Optimizing Long-term Cloud Costs by Mixing VM Purchasing Options”. In: *2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2020.

[44] Roland Mathá et al. “Simplified Workflow Simulation on Clouds based on Computation and Communication Noisiness”. In: *IEEE Transactions on Parallel and Distributed Systems* 31.7 (2020), pp. 1559–1574.

[45] Benay Ray et al. “Proactive Fault-Tolerance Technique to Enhance Reliability of Cloud Service in Cloud Federation Environment”. In: *IEEE Transactions on Cloud Computing* (2020).

[46] *Amazon EBS features*. https://aws.amazon.com/ebs/features/. [Online; accessed 29-October-2020].

[47] *Amazon EBS Volume Types*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html. [Online; accessed 01-May-2020].

[48] *Amazon EC2 Pricing*. https://aws.amazon.com/ec2/pricing/on-demand/. [Online; accessed 01-May-2020].

[49] *Amazon Elastic Compute Cloud*. https://aws.amazon.com/ec2/. [Online; accessed: 01-May-2020].

[50] *Attaching a volume to multiple instances with Amazon EBS Multi-Attach*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes-multi.html. [Online; accessed 21-August-2020].

[51] Chris Braun. *TrafficToll*. https://github.com/cryzed/TrafficToll. [Online; accessed 01-May-2020].

[52] *cgroups – Linux control groups*. http://man7.org/linux/man-pages/man7/cgroups.7.html. [Online; accessed 01-May-2020].

[53] *CPU Credits and Baseline Performance for Burstable Performance Instances*. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html. [Online; accessed 01-May-2020].

[54] *Google Compute Engine*. https://cloud.google.com/compute. [Online; accessed: 01-May-2020].

[55] Bert Hubert, Jacco Geul, and Simon Sehier. *The Wonder Shaper 1.4*. https://github.com/magnific0/wondershaper. [Online; accessed 01-May-2020].

[56] *iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool*. https://github.com/esnet/iperf. [Online; accessed 01-May-2020].

[57] Alexey Kopytov. *SysBench: a system performance benchmark*. https://github.com/akopytov/sysbench. [Online; accessed 01-May-2020].

[58] Angelo Marletta. *CPU usage limiter for Linux*. https://github.com/opsengine/cpulimit. [Online; accessed 01-May-2020].

[59] *Microsoft Azure*. https://azure.microsoft.com/. [Online; accessed: 01-May-2020].

[60] *N1 shared-core machine types*. https://cloud.google.com/compute/docs/machine-types#n1_shared-core_machine_types. [Online; accessed 15-August-2020].

[61] *nice - run a program with modified scheduling priority*. https://linux.die.net/man/1/nice. [Online; accessed 01-May-2020].

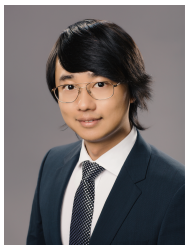
[62] Corey Sanders. *Introducing B-Series, our new burstable VM size*. https://azure.microsoft.com/en-us/blog/introducing-b-series-our-new-burstable-vm-size/. [Online; accessed 15-August-2020].

[63] *tc - show / manipulate traffic control settings*. https://linux.die.net/man/8/tc. [Online; accessed 01-May-2020].

[64] Sage Weil. *New in Luminous: BlueStore*. https://ceph.com/community/new-luminous-bluestore/. [Online; accessed 01-May-2020].

Riccardo Pinciroli received the M.S. and Ph.D. degrees in computer engineering from Politecnico di Milano, in 2014 and 2018, respectively. He is currently a Postdoctoral Fellow in Computer Science at GSSI – Gran Sasso Science Institute. His research interests include stochastic modeling, performance evaluation, energy efficiency, and uncertainty propagation applied to cloud computing, data-centers, cyber-physical systems, and IoT environments.

Ahsan Ali is a Ph.D. student in the Department of Computer Science and Engineering at the University of Nevada, Reno. He earned his B.S. in Telecommunication Engineering from NUCES in Islamabad, Pakistan and M.S. degrees in Computer Science from KOC University in Istanbul, Turkey in 2012 and 2016, respectively. His research interests include machine learning, cloud/serverless computing, performance evaluation, and HPC.



Feng Yan is an Assistant Professor in the Department of Computer Science and Engineering at the University of Nevada, Reno. His research includes big data, machine learning, cloud/edge/fog computing, HPC, storage, and cross-disciplinary topics. He obtained the B.S degree in Computer Science from Northeastern University, and the M.S. (2011) and Ph.D. (2016) degrees in Computer Science from William and Mary. He received the Best Paper Award at CLOUD 2019 and the Best Student Paper Award at IEEE CLOUD 2018.

Evgenia Smirni is the Sidney P. Chockley professor of computer science at William and Mary, Williamsburg, VA. Her research interests include queuing networks, stochastic modeling, resource allocation, storage systems, cloud computing, workload characterization, and modeling of distributed systems and applications. She is an ACM Distinguished scientist and an IEEE Fellow.