It's not a Sprint, it's a Marathon: Stretching Multi-resource Burstable Performance in Public Clouds (industry track)

Ahsan Ali
aali@nevada.unr.edu
University of Nevada, Reno
Reno, NV, USA

Riccardo Pinciroli
rpinciroli@wm.edu
William and Mary
Williamsburg, VA, USA

Feng Yan
fyan@unr.edu
University of Nevada, Reno
Reno, NV, USA

Evgenia Smirni
esmirni@cs.wm.edu
William and Mary
Williamsburg, VA, USA

ABSTRACT

During the past few years, all leading cloud providers introduced *burstable instances* that can sprint their performance for a limited period to address sudden workload variations. Despite the availability of burstable instances, there is no clear understanding of how to minimize the waste of resources by regulating their burst capacity to the workload requirements. This is especially true when it comes to non-CPU-intensive applications. In this paper, we investigate how to limit network and I/O usage to optimize the efficiency of the bursting process. We also study which resource shall be controlled to benefit both cloud providers and end-users. We design MRburst (Multi-Resource burstable performance scheduler) to automatically limit multiple resources (i.e., network, I/O, and CPU) and make the application comply with a user-defined service level objective (SLO) while minimizing wasted resources. MRburst is evaluated on Amazon EC2 using two multi-resource applications: an FTP server and a Ceph system. Experimental results show that MRburst outperforms state-of-the-art approaches by allowing instances to speed up their performance for up to 2.4 times longer period while meeting SLO.

KEYWORDS

Burstable Instances, Resource Management, Quantile Regression

ACM Reference Format:
Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2019. It's not a Sprint, it's a Marathon: Stretching Multi-resource Burstable Performance in Public Clouds (industry track). In *20th International Middleware Conference Industrial Track (Middleware Industry '19), December 9–13, 2019, Davis, CA, USA.* ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION

All major cloud providers have by now introduced burstable instances to improve spare resource utilization. Burstable instances come with initial credits for each type of resources. These credits are used to temporally sprint the performance of instance beyond the baseline point of reference. Cloud service providers typically disclose some information about the resource credits. For example, Amazon Web Services (AWS) explicitly mentions the CPU and I/O initial number of credits and the generation/consumption rate, but does not provide any information regarding the network credits. This initial transient period in which an instance uses up its initial credits and essentially operates above baseline performance is called *Credit Depletion Period*. Any reduction in resource utilization during the instance execution decreases the credit consumption rate and extends the credit depletion period. For example, one CPU credit of an AWS *t2.micro* instance is consumed in 2 minutes if the CPU is 50% utilized or in 1 minute when the CPU utilization is 100%. After consuming all initial credits, the instance operates at its baseline performance. Periodically, it generates new credits with a rate that is commensurate to its size, i.e., credit generation is capped. In general, the credit generation rate is equal to the number of credits that the instance requires for working with baseline performance. Hence, the number of resource credits does not change when the instance operates at baseline performance.

Our thesis is that depending on the user performance objectives, it may be beneficial to stretch the instance credit depletion period, by slowing down the consumption of its initial credits. This implies that if users only require a certain performance level (i.e., meet SLO) during the initial transient period, then the period when the instance operates above baseline performance can be stretched significantly. Even more, if users can *predict* when the credit is completely depleted, the application can be migrated to a fresh instance with full credits and enjoy a new extended period of higher-than-baseline performance with significantly lower cost. It is worth to point out that the above also benefits the cloud service providers in the long run as the burstable mechanism is "regulated" using spare resources that are otherwise wasted. Regulating the resource utilization can provide a finer-grained spare resource control to further improve the utilization of spare resources.

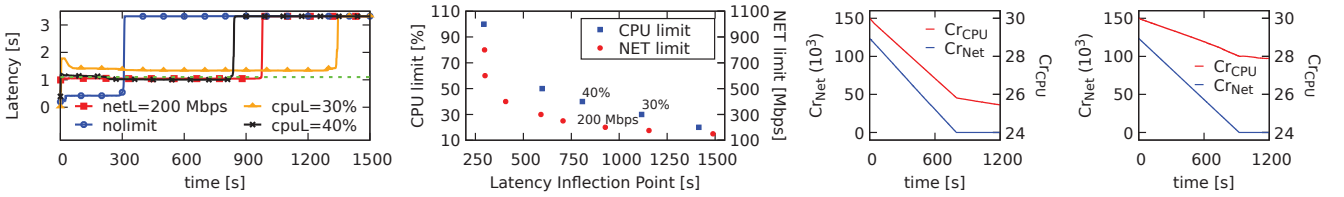
To slowdown credit consumption, a limit on resource usage must be enforced. Initial characterization experiments on AWS confirm that limiting the performance of an instance extends its credit depletion period. While this idea of limiting resources by throttling their consumption in a judicious way is appealing, its challenges are multi-fold: *i)* there is nebulous information about credit generation and consumption mechanisms of the instance; *ii)* expensive profiling is required to analyze the ample search space of resource limitations; *iii)* there may be unexpected interactions of different resources and their performance effect on each other, e.g., limiting CPU utilization leads to reducing network bandwidth and vice versa.

This paper presents *Multi-Resource burstable performance scheduler* (MRburst), a framework that improves the long term performance of an instance by analyzing multiple resources (i.e., CPU, I/O, and network) to perform throttling and bypasses the above challenges in a black-box manner. MRburst extends the credit depletion period

Middleware Industry '19, December 9–13, 2019, Davis, CA, USA

ACM ISBN 978-1-4503-7009-7/19/12…$15.00

DOI: 10.1145/3366626.3368130



(a) 95th-percentile latency **(b) Limits vs. Latency inflection point** **(c) Credits, CPU limit** = 40% **(d) Credits, NET limit** = 200 **Mbps**

Figure 1: Advantages in throttling and monitoring other resources in addition to CPU when Ceph is deployed on AWS.

while complying with application SLOs (e.g., pre-defined percentile latencies) and schedules migration before performance degrades. Specifically, MRburst samples a few *limited* observations by profiling the application 95th-percentile latency with different resource limitations. The optimal resource throttling is selected to reduce credit consumption and maintain the 95th-percentile latency (or any predefined percentile) below the agreed SLO. MRburst adopts CPU, I/O, and network credit depletion models to predict the performance degradation of the instances due to credit exhaustion.

We prototype MRburst and evaluate it on Amazon EC2. We consider two applications, an FTP server (to transfer files from servers to clients) and a Ceph system (for object storage [15]). The former is a network-intensive application that may consume I/O and network credits. The latter requires CPU, I/O, and network credits. Experiments show that MRburst outperform available strategies by extending the credit depletion period of an instance up to 2.4 times and halving the number of required migrations (for the purpose of extending the burstable time period. To the best of our knowledge this is the first attempt to optimize multi-resource applications exploiting network and I/O bursting performance.

2 MOTIVATION

Previous work [1, 17] shows that it is possible to extend the CPU credit depletion period by limiting CPU usage through *cpulimit*, but such work focuses on CPU-intensive applications only. Real-world applications can easily stress resources other than CPU, e.g., they may be network- or I/O-intensive.

Similar to CPU, network and I/O performance are governed by a credit mechanism, that allows resources to burst their performance beyond the baseline. Resource throttling allows for decreasing credit consumption rate and for extending the credit depletion period. Tools exist for limiting I/O, network, and CPU performance, including *cgroups* [7], *Wonder Shaper* [3], and *cpulimit* [8], respectively. When the system runs out of credits, it can only operate with baseline performance. Thus, it is crucial to keep track of the credit consumption rate of all resources to make system performance comply with the user-defined SLO. Figure 1(a) shows the effect of limiting CPU or network on a Ceph cluster with 200 users. Ceph [15] is a storage platform that implements distributed object storage while providing performance and dependability. When no limitation is applied, Ceph uses all available resources to process the incoming requests and depletes its resource credits too fast: it starts violating the SLO after only 300 seconds. Limiting either CPU (i.e., 40%) or network (i.e., 200 Mbps) allows extending the credit depletion period up to 900 seconds. In this case, Ceph performance is slightly worse, but its latency is still shorter than the user-defined SLO (see the green dashed line). The credit depletion period can be further extended by using stricter limitations (e.g., *cpulimit* = 30%), but now SLO is violated.

Although both limitations (i.e., CPU and network) in Figure 1(a) allow Ceph to operate with the same latency (i.e., 1 second), network limitation makes the credit depletion period (CDP) 120 seconds longer. The different credit depletion periods when throttling different resources depend on the system environment (e.g., application, system configuration). Hence, it is nontrivial to determine which limitation is more efficient apriori.

Figure 1(b) depicts CPU and network limit as a function of the latency inflection point, i.e., the credit depletion period. It shows that the credit depletion period is not enough to identify the best limitation to apply since no performance measures are available. One must analyze a considerable search space (i.e., limitation types, limitation values, workload, and latency) to determine how to throttle the system. Network and CPU limits considered in Figure 1(a) are highlighted in Figure 1(b) for easier comparison.

Figures 1(c) and 1(d) depict the available network (left y-axis) and CPU (right y-axis) credits for two cases plotted in Figure 1(a) (i.e., *cpulimit* = 40% and *Wonder Shaper* = 200 Mbps). Although such limitations provide the same performance, resource credit consumption rate, i.e., resource usage, vary based on the applied limitation. For example, after 1200 seconds, there are 25 CPU credits if *cpulimit* = 40% is applied or 28 CPU credits when *Wonder Shaper* = 200 Mbps is used. Similar considerations apply to network credits. This means that there is a relationship among resource usage, but it may be hard to determine its nature in advance.

Non-CPU-intensive applications generally consume network or I/O credits faster than CPU-intensive ones. Hence, monitoring only CPU credit consumption is inadequate to migrate an instance before its performance deterioration. For example, Ceph starts operating with baseline performance after consuming all network credits of the instance (see Figures 1(c) and 1(d)). Independently of the adopted limitation, the instance still has almost all its CPU credits when network ones are exhausted. For this reason, to efficiently schedule instance migration, it is necessary to monitor all resources to determine which one is the first one to exhaust its credits.

To summarize: *i)* real-world applications need different resources (i.e., CPU, I/O, and network) to work correctly; *ii)* it is hard to determine in advance which limitation must be applied to optimize resource usage; *iii)* it is necessary to profile an enormous search space for determining the most efficient limitation; *iv)* interdependence among resources is not definable apriori; *v)* resources consume their credits with different rates. Therefore, it is crucial to have a framework to bypass all of the above challenges and provide seamless answers to the following. *i)* Is it possible to control non-CPU-intensive applications and extend the credit depletion period of the hosting instance while complying with user-defined SLOs? *ii)* Which limitation type is the most efficient to apply? *iii)* Which resource consumes up its credits first? To address the above challenges

Table 1: Parameters of the optimization problem in Eq. (1) for an AWS *t2.micro* instance. *vs* stands for *volume size* in GiB.

	$C_k(0)$ [cr]	C_k^{max} [cr]	R_c^k	R_e^k
k = CPU	30	144	1 cr/min	0.1 cr/min
k = I/O	5.4×10^6	5.4×10^6	1 cr/IO	$3 \cdot vs$ cr/sec
k = net	123,617	123,617	1 cr/Mbit	61 cr/sec

we propose MRburst, a black-box approach that optimizes resource usage by identifying the most effective resource to throttle and the most profitable throttling level. It also monitors all resources to migrate the application before performance degradation is observed. MRburst benefits also cloud providers by optimizing resource usage and decreasing the number of launched instances. In Figure 1(a), performance degrades after 300 seconds when no resource is throttled. A new instance is required for migration every 5 minutes to meet the SLO. Resource throttling (e.g., *Wonder Shaper* = 200 Mbps) extends the credit depletion period to 900 seconds and the application can be hosted by a new VM every 15 minutes.

3 METHODOLOGY

Given a workload, MRburst finds the best limitation that maximizes spare resources while complying with the advertised SLO. For this purpose, MRburst uses empirical measurements (lightweight profiling) and an analytical method (quantile regression). MRburst enables control over CPU, network, and I/O by integrating *cpulimit*, *Wonder Shaper*, and *cgroups*:

- *cpulimit* [8] is a tool that throttles the CPU utilization of a process by pausing the process itself to keep its CPU usage under a defined maximum value;
- *Wonder Shaper* [3] works with *Linux Traffic Control* [2] and allows users to limit network bandwidth by specifying a maximum number of bits that can be transferred every second over a specific network interface;
- *cgroups* [7] allows users to specify the maximum number of input/output operations (IOPS) that can be executed every second on an I/O device.

MRburst monitors credits availability of each resource to efficiently schedule instance migration before credit depletion. Since MRburst can throttle any resource, it provides users with the *best* resource to be throttled based on the target SLO. MRburst is composed of four main components: *Lightweight Profiler*, *Prediction Model*, *Scheduler*, and *Migration Planner*. They are described in the following.

3.1 Lightweight Profiler

Exhaustive profiling of system performance is the main way to maximize the efficiency of spare resources while meeting SLOs. Such a strategy is costly and time-consuming due to the enormous amount of data and measurements required. For example, assuming C different values of *cpulimit* and L different workloads, one must run $C \cdot L$ experiments for profiling. It is necessary to collect a large number of samples for each experiment to determine percentile latency correctly, especially for high percentile values. If t_C is the time required to reach statistical stability in profiling, exhaustive profiling for *cpulimit* will require $C \cdot L \cdot t_C$ time units to be completed. One must execute the same profiling procedure also for other resource limitations, i.e., network (*Wonder Shaper*) and I/O (*cgroups*).

MRburst adopts a lightweight profiling strategy. First, it conducts some sparse sampling with experiments with different values for *cpulimit* C, *Wonder Shaper* W, *cgroups* D, and workload L. Then, it adopts an analytical model to determine the missing values. Assume that the percentage of data collected for C, W, D, and L is σ, ω, δ, and λ, respectively. The lightweight strategy profiling time is $\lambda \cdot L \cdot \left(\sigma \cdot C \cdot t_C + \omega \cdot W \cdot t_W + \delta \cdot D \cdot t_D\right)$, i.e.,

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

times shorter than the exhaustive strategy.

3.2 Prediction Model

We propose a prediction methodology that uses profiling data, SLO, and current workload to determine a value of *Wonder Shaper*, *cpulimit*, or *cgroups* which allows the system to comply with the given SLO and maximize the efficiency of spare resources.

3.2.1 Problem Formulation. Since available credits determine spare resources, we define *Credit Efficiency* of a resource k as the distance between its average credit depletion time, T_d^k, and the instance migration time, T_m, under a given SLO constraint. Hence, to maximize credit efficiency, we define the following optimization problem:

$$\begin{aligned} &\text{maximize} && T_d^k - T_m,\ k \in \{\text{CPU, I/O, net}\} \\ &\text{subject to} && P_i \leq \text{SLO}, \end{aligned} \tag{1}$$

where P_i is the ith-percentile latency and it depends on CPU usage, IOPS, and network bandwidth. T_m includes the time required to migrate an instance and depends on the migration strategy. Given a resource k (i.e., CPU, I/O, or network), T_d^k is its credit depletion time and it is computed by:

$$T_d^k = \frac{C_k(t)}{R_c^k \cdot M_k - R_e^k}, \tag{2}$$

where $C_k(t)$ is the number of credits available at time t, R_e^k is its credit earning rate, R_c^k is its credit consumption rate, and M_k is the performance index that determines its usage (i.e., utilization for CPU and throughput for I/O and network). AWS provides $C_k(0)$, R_e^k, R_c^k, and the model itself for CPU and I/O [10, 11], but it does not divulge any network parameters and its credit depletion model.

We determine the missing input parameters of the optimization problem by benchmarking the network performance. The initial number of network credits, $C_{net}(0)$, is derived from Eq. (2), where R_e^{net} is the network performance baseline (i.e., 61 credits per second for a *t2.micro*), X_{net} is the network bandwidth and it depends on the limitation imposed through *Wonder Shaper*, and R_c^{net} is observed to be 1 credit per Mbit. T_d^{net} is the time when system performance deteriorates. Table 1 shows the parameters of the optimization problem for an AWS *t2.micro* instance. Note that each CPU credit enables the instance to use the CPU with 100% utilization for one minute, an I/O credit allows the application to execute one input/output operation (i.e., read or write), and a network credit provides the VM with 1 Mbps network bandwidth. The output of the optimization problem in Eq. (1) is the limitation value for *cpulimit*, *cgroups*, or *Wonder Shaper* that maximizes the credit efficiency without violating SLOs.

Table 2: Network credit depletion periods estimated through Eq. (2) and compared to the measured results from AWS.

App.	Limited Res.	Limit	T_d^{estim}	T_d^{real}	Error
FTP	Network	250 Mbps	757 s	765 s	1.04%
FTP	I/O	300 reads/s	497 s	521 s	4.40%
Ceph	Network	400 Mbps	399 s	401 s	0.50%
Ceph	CPU	30%	286 s	296 s	3.38%

3.2.2 Analytical Model. In order to determine the optimal limitation value that maximizes credit efficiency and does not violate SLO, we evaluate the *i*th-percentile latency, P_i, for different resource limitations and workloads. We use quantile regression [5] to derive an analytical model of the application. Quantile regression is a statistical inference method that can estimate and extrapolate the relationship between conditional quantile functions. The robustness to non-normal errors and outliers – compared to linear regression – and the absence of any assumption regarding the underlying distribution of the data [18] make quantile regression suitable for the purpose of MRburst.

Quantile regression admits input data of one-dimensional samples. Latency prediction takes into consideration resource limitations and loads, which form three two-dimensional spaces (one for each resource). Focusing on each limitation type (i.e., *cpulimit*, *Wonder Shaper*, and *cgroups*), we adjust the quantile regression presented in [5] by first training the model assuming a constant load and varying the value of the limitation. Then, we fix the limitation value and vary the load. Finally, we derive a global model for each limitation by combining the models trained in the two steps.

3.3 Scheduler

The scheduler analyzes the current workload of the system, then it sets *cpulimit*, *cgroups*, or *Wonder Shaper* to limit resource consumption while complying with the SLO. Specifically, the scheduler initially adopts the smallest limitation such that the *i*th-percentile latency is less than the SLO. For dynamic workloads, the scheduler must adjust the resource limitation every time the workload changes. The scheduler continuously monitors the system in order to adjust the throttling values to the changing workload.

3.4 Migration Planner

To correctly schedule an instance migration, MRburst monitors CPU, I/O, and network credits to predict when the performance of the instance is degrading. MRburst keeps tracking of resource usage and it uses Eq. (2) to determine the credit depletion period of all resources. Then, the migration planner compares the available credits with thresholds that identify the minimum number of credits required for smooth operation of the application. The application is migrated when at least one resource has less credits than its threshold.

4 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of MRburst through experiments on Amazon EC2. We study the accuracy of the network prediction model presented in Section 3, determine the MRburst prediction error, test its performance when it manages dynamic workloads, and compare MRburst with exsiting strategies.

4.1 System overview

MRburst is evaluated with Amazon EC2 *t2.micro* instances, that have 1 vCPU and 1 GB memory. All instances are created with Ubuntu Server 18.04 LTS in the *us-east-1* region and each instance uses the same Amazon Virtual Private Cloud, i.e., the same subnet.

We consider two applications: FTP and Ceph. The FTP server is placed on a *t2.micro* instance, while the client is on a *m5.large* since that type of instance is not controlled by credits and its performance is not affected by burstable resources. The client can vary the system workload by adjusting the number of concurrent users downloading a 10 MB file from the FTP server. A 40 GB gp2 volume (i.e., 120 IOPS/s baseline) is attached to the FTP server. Caches are disabled and users need to access the I/O to download the file, so a request consumes both I/O and network credits.

The Ceph cluster consists of a Monitor/Client and an Object Storage Device (OSD) deployed on *m5.large* and *t2.micro* instances, respectively. The cluster stores data in one pool and does not implement object replication since it is composed by only one OSD. The OSD mounts a 100 GB gp2 volume (i.e., 300 IOPS/s baseline). Such a system consumes credits of all resources (i.e., CPU, I/O, and network). Since I/O usage depends on the OSD's file system and its storage back-end (i.e., BlueStore [14]), it is nontrivial to throttle I/O performance through *cgroups*. For this reason, we only analyze the effect of CPU and network on system performance. The OSD size is large enough to make the I/O credit depletion period the longest one.

4.2 Workload

We evaluate MRburst with static and dynamic loads. The number of users concurrently reaching the server defines the system workload.

Static workload: The number of users in the system is fixed. These basic experiments allow training the quantile regression model and study its accuracy for a steady workload.

Dynamic workload: After training the regression model, we evaluate MRburst under a fluctuating workload. This allows analyzing its ability to adjust the throttling parameters of the three resoruces when the number of users varies over time for a given SLO.

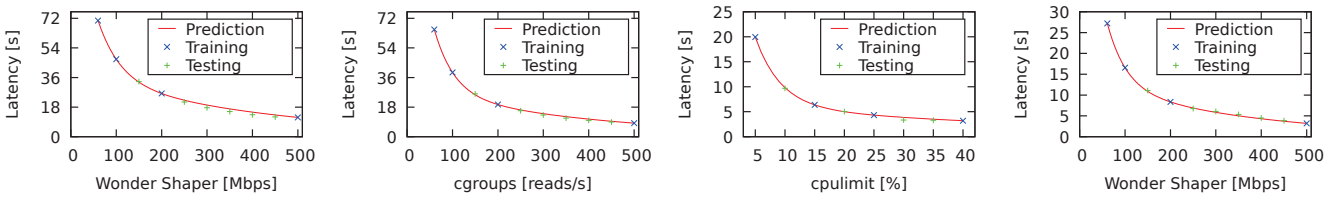
4.3 Network Credit Depletion Period: Analytic Model vs. Experimental Results

The network credit depletion model given in Eq. (2) accurately estimates the network credit depletion period. To show that, we run experiments for FTP and Ceph with different CPU, I/O, and network limitations. The depletion time estimated by the model is compared to the performance deterioration time observed in the experiments.

Table 2 shows the error of the model for different applications and limitation types and values. It is computed by: $|T_d^{estim} - T_d^{real}| / T_d^{real}$. In general, the accuracy of the prediction model is high, especially when *Wonder Shaper* is used, reaching errors of around 1%. We point out that the CPU and I/O credit depletion models are provided by AWS [10, 11].

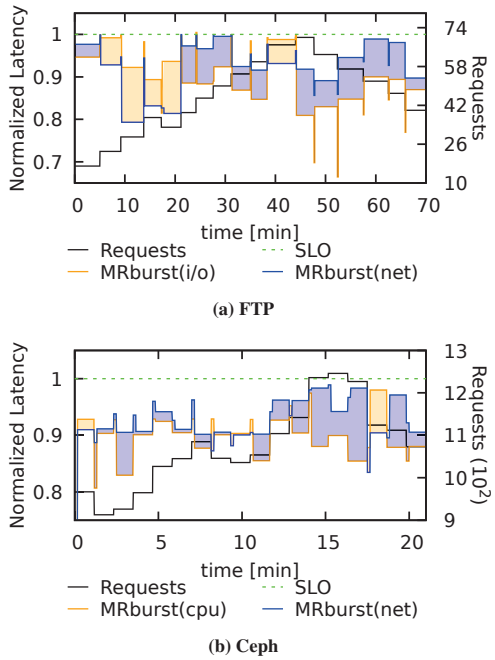
4.4 MRburst with Static Workload

For each system configuration (i.e., type of application, workload, and limitation), we analyze the prediction model derived by MRburst and compute its mean absolute percentage error as: $|Y(l) - y_l| / y_l$,



(a) FTP, Users = 50, *Wonder Shaper* **(b) FTP, Users = 50, *cgroups*** **(c) Ceph, Users = 1600, *cpulimit*** **(d) Ceph, Users = 1600, *Wonder Shaper***

Figure 2: 95th-percentile latency prediction for FTP (mean error $< 5\%$) and Ceph (mean error $< 10\%$) with static workload.



(a) FTP

(b) Ceph

Figure 3: 95th-percentile latency of FTP and Ceph with fluctuating workload. Latencies are normalized over the SLO (simple ratio). Concurrent requests represent the system workload.

where $Y(l)$ and y_l are the predicted and measured 95th-percentile latencies for throttling parameters l, respectively.

The prediction model for FTP is trained limiting the network to $\{61, 100, 200, 500\}$ Mbps and the I/O to $\{61, 100, 200, 500\}$ IOPS. The number of users in the system is $N = 50$ and each experiment lasts 300 seconds. The predicted latency follows the same trend of the observed one (Figures 2(a) and 2(b)) independently of the applied limitation (i.e., *Wonder Shaper* or *cgroups*, respectively) and the mean absolute error is not larger than 5%. Similar results are observed when latency is plotted against the workload and the throttling value is fixed.

When applied to Ceph (Figures 2(c) and 2(d)) MRburst increases the credit efficiency of the instances by throttling either CPU or network. The prediction model is trained for *cpulimit* = $\{5, 15, 25, 40\}$% and *Wonder Shaper* = $\{61, 100, 200, 500\}$ Mbps, each experiment lasts 200 seconds, and $N = 1600$ users. The model performs better if MRburst limits the network to increase the credit efficiency, but its error is smaller than 10% independent of the throttled resource.

These experiments show that MRburst's prediction model is consistently robust in predicting system latency. The prediction model can handle different applications and it works with various loads and resource limitations. Limiting network is as good as throttling I/O, but it is generally more accurate than *cpulimit*.

4.5 MRburst with Dynamic Workload

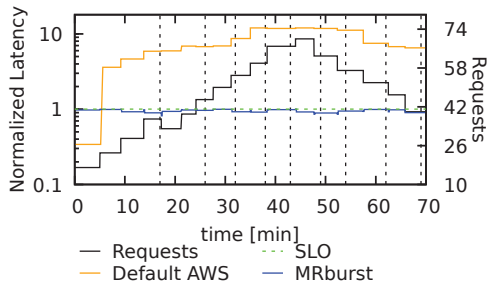
We illustrate how to use the prediction model for selecting the optimal limitation value on the fly, i.e., when the number of users connected to the system changes. MRburst minimize the amount of wasted resources without sacrificing the target application latency. The closer the response time is to the SLO, the more efficient the framework. When the number of users into the system changes, the framework identifies the new workload during the *observation window* which we set to 10 seconds. Results are presented for a fixed SLO, but experiments can be generalized for varying objectives.

MRburst automatically adopts the limitation type that maximizes the instance credit efficiency without violating the SLO. Hence, the credit depletion period and the time to next migration are extended. Before throttling the system performance with the chosen limitation, MRburst evaluates how limiting each resource (i.e., CPU, I/O, and network) affects the application. To highlight MRburst's capability in selecting the optimal throttling, we analyze the effect of MRburst on system performance when it limits one resource at a time.

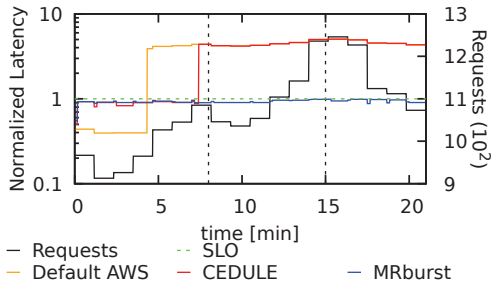
First, MRburst performance is evaluated with FTP when the workload changes dynamically. The number of concurrent requests varies from 10 to 100 and the time duration of each load is between 180 and 350 seconds. Figure 3(a) shows the application 95th-percentile latency when MRburst adopts *Wonder Shaper* (blue line) or *cgroups* (yellow line), while the workload varies (black line). Latencies are normalized over the SLO and are plotted against the left y-axis, while the system workload is plotted against the right y-axis. The system runs out of network credits sooner than of I/O ones and MRburst migrates the FTP server to a new VM in order to meet the SLO requirement. MRburst complies with the given SLO independently of the resource that is throttled, notice that the latencies are always below the objective line. The area between the two latency lines represents the resource capacity saved by each strategy. If the area is blue, then network throttling provides a greater credit efficiency. Otherwise, I/O throttling is preferable (yellow area). The FTP migration time is shown by vertical dashed lines in Figure 4(a).

We also test MRburst using Ceph and vary the application load from 913 to 1246 concurrent requests. The number of users connected to the system varies every 70 seconds. Figure 3(b) depicts the application latency when MRburst throttles either network or CPU. With MRburst, Ceph always complies with the given SLO, but it works better when limiting the network bandwidth. Ceph migration time is shown in Figure 4(b).

Experiments with dynamic workloads show that MRburst can efficiently adapt to load variations. They exhibit that limiting network is generally equivalent to limiting I/O and more efficient than limiting CPU. MRburst can select the limitation that better applies to



(a) FTP



(b) Ceph

Figure 4: MRburst compared to state-of-the-art approaches. Latencies are normalized over the SLO (simple ratio). Concurrent requests represent the system workload. Vertical dashed lines point out MRburst migration times.

the current application and system workload, therefore maximizing the system credit efficiency.

4.6 Improvements to other strategies

Figures 4(a) and 4(b) compare MRburst to CEDULE [1] and the default AWS strategy to show MRburst's ability to increase credit efficiency by extending the credit depletion period. 95th-percentile latencies (left y-axis) are normalized over the SLO and the workload is plotted against the right y-axis. For both applications, the default AWS strategy does not use extra resources efficiently and it violates the SLO after a few minutes. Indeed, it greedily uses all the available credits and does not autonomously migrate the application when the SLO is not met. CEDULE can monitor and throttle only CPU and it improves credit efficiency only when CPU is involved in request processing (i.e., Ceph), otherwise it performs as the default strategy (i.e., FTP). CEDULE does not monitor available network credits and cannot migrate the application before performance deterioration. MRburst always complies with the SLO by efficiently using the spare resources and monitoring CPU, I/O, and network available credits for migrating the application before SLO is violated (see vertical dashed lines). In the considered cases, MRburst makes the credit depletion period up to 240% and 95% longer for FTP and Ceph, respectively, with respect to the default AWS mechanism. Increased credit efficiency benefits cloud providers by reducing wasted resources and diminishing the number of new instances required to meet the SLO. MRburst requires starting a new instance 1.5 times less frequently than the default AWS mechanism for FTP. Similarly, required new instances for Ceph are halved when using MRburst instead of the default mechanism.

5 RELATED WORK

Amazon Web Services started offering burstable instances in 2010 with *t1.micro* instances. Since 2014, all leading cloud providers offer the same type of low-cost instances for applications with low traffic and throughput. Prior work investigates this type of VMs to identify the best practices to get advantages from *burstable* features, mainly focusing only on CPU performance. Wen et al. [16] statistically analyze T1 instances (i.e., the first generation of AWS burstable instances) and propose to inject delays to optimize performance and cost of *t1.micro* VMs. Jiang et al. [4] investigate T2 instances (i.e., the AWS second generation burstable instances) and propose an analytical performance model to study burstable instances given the type and configuration of a VM. For this purpose, they consider only CPU performance and the available CPU credits. Their model can be used to find which instance type a tenant should select to get the best trade-off between cost and performance. Several authors focus on managing credits in T2 instances to improve VM performance. For example, Leitner and Scheuner [6] propose a basic model to analyze T2 instances and investigate boosting performance by restarting instances when all credits are depleted. Unfortunately, such a practice is no more functional since AWS has introduced constraints on rebooting T2 instances [9]. The lifetime of CPU credits is extended in [17] by using *cpulimit*. That allows surpassing the performance of the delay strategy proposed in [16]. *cpulimit* is used by CEDULE [1] to automatically throttle CPU and extend its credit depletion period while meeting user-determined SLOs for applications with multi-instance dependency (e.g., TPC-W [12]). Wang et al. [13] point out that the mechanism regulating the credits consumption of network and I/O follows a token-bucket like model but they do not provide any strategy to optimize the credit consumption rate of network and I/O, they provide just some observations. To the best of our knowledge, MRburst is the first framework that operates on CPU, network, and I/O in a unified manner to optimize both the performance and cost of burstable instances in a cloud environment.

6 CONCLUDING REMARKS

This paper presents MRburst, a scheduling framework for multi-resource burstable instances, that increases the efficiency of resources. MRburst uses lightweight profiling and quantile regression to automatically sets the best limitation that reduces the number of credits consumed without violating the given SLO.

The efficiency of MRburst is tested on AWS with non-CPU-intensive applications (i.e., FTP and Ceph). The prediction model is consistently accurate and its distance from the observed latency is always smaller than 10%. MRburst extends the credit depletion period up to 2.4 times with respect to other strategies and benefits cloud providers by increasing resource usage.

In the future, we aim to extend MRburst with instance type recommendation. This way, it can also select the best T2 instance to host the application, based on its workload and SLO. In order to minimize the system cost, instance recommendation will account for the cost of each instance and the number of migrations expected to complete a job.

Acknowledgements: This work is supported by NSF grants CCF-1218758, CCF-1649087, CCF-1756013, and IIS-1838024.

REFERENCES

[1] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2018. CEDULE: A Scheduling Framework for Burstable Performance in Cloud Computing. In *2018 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 141–150.

[2] Bert Hubert. 2006. tc - show / manipulate traffic control settings. (2006). https://linux.die.net/man/8/tc [Online; accessed 23-May-2019].

[3] Bert Hubert, Jacco Geul, and Simon Sehier. 2017. The Wonder Shaper 1.4. (2017). https://github.com/magnific0/wondershaper [Online; accessed 24-May-2019].

[4] Yuxuan Jiang, Mohammad Shahrad, David Wentzlaff, Danny HK Tsang, and Carlee Joe-Wong. 2019. Burstable instances for clouds: Performance modeling, equilibrium analysis, and revenue maximization. In *Proceedings-IEEE INFOCOM*.

[5] Roger Koenker and Kevin F Hallock. 2001. Quantile regression. *Journal of economic perspectives* 15, 4 (2001), 143–156.

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

[7] David MacKenzie and James Youngman. 2010. cgroups – Linux control groups. (2010). https://linux.die.net/man/1%2Fgroups [Online; accessed 19-May-2019].

[8] Angelo Marletta. 2012. CPU usage limiter for Linux. (2012). https://github.com/opsengine/cpulimit [Online; accessed 24-May-2019].

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

[10] Amazon Web Services. 2016. Amazon EBS Volume Types. (2016). https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html [Online; accessed 19-May-2019].

[11] Amazon Web Services. 2017. CPU Credits and Baseline Performance for Burstable Performance Instances. (2017). https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html [Online; accessed 22-May-2019].

[12] Wayne D Smith. 2000. TPC-W: Benchmarking an ecommerce solution. (2000).

[13] Cheng Wang, Bhuvan Urgaonkar, Neda Nasiriani, and George Kesidis. 2017. Using Burstable Instances in the Public Cloud: Why, When and How? *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 1 (2017), 11.

[14] Sage Weil. 2017. New in Luminous: BlueStore. (2017). https://ceph.com/community/new-luminous-bluestore/ [Online; accessed 25-April-2019].

[15] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 307–320.

[16] Jiawei Wen, Lei Lu, Giuliano Casale, and Evgenia Smirni. 2015. Less can be more: Micro-managing vms in amazon ec2. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 317–324.

[17] Feng Yan, Lihua Ren, Daniel J Dubois, Giuliano Casale, Jiawei Wen, and Evgenia Smirni. 2017. How to Supercharge the Amazon T2: Observations and Suggestions. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*. IEEE, 278–285.

[18] Yunqi Zhang, David Meisner, Jason Mars, and Lingjia Tang. 2016. Treadmill: Attributing the source of tail latency through precise load testing and statistical inference. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 456–468.