Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Dynamic Resource Allocation in the Cloud with Near-Optimal Efficiency

Sebastian Perez-Salazar, Ishai Menache, Mohit Singh, Alejandro Toriello

informs
http://pubsonline.informs.org/journal/opre
OPERATIONS RESEARCH
Articles in Advance, pp. 1–21
ISSN 0030-364X (print), ISSN 1526-5463 (online)

Methods

Dynamic Resource Allocation in the Cloud with Near-Optimal Efficiency

Sebastian Perez-Salazar,[a] Ishai Menache,[b] Mohit Singh,[a] Alejandro Toriello[a]

[a] Georgia Institute of Technology, Atlanta, Georgia 30332; [b] Microsoft Research, Redmond, Washington 98052

Contact: sperez@gatech.edu, https://orcid.org/0000-0003-4534-7721 (SP-S); ishai@microsoft.com, https://orcid.org/0000-0002-2540-236X (IM); mohit.singh@isye.gatech.edu, https://orcid.org/0000-0002-0827-233X (MS); atoriello@isye.gatech.edu, https://orcid.org/0000-0002-3147-0764 (AT)

Received: August 15, 2019
Revised: September 16, 2020
Accepted: January 21, 2021
Published Online in Articles in Advance: October 29, 2021

Subject Classification: analysis of algorithms: suboptimal algorithms
Area of Review: Optimization

https://doi.org/10.1287/opre.2021.2138

Abstract. Cloud computing has motivated renewed interest in resource allocation problems with new consumption models. A common goal is to share a resource, such as CPU or I/O bandwidth, among distinct users with different demand patterns as well as different quality of service requirements. To ensure these service requirements, cloud offerings often come with a service level agreement (SLA) between the provider and the users. A SLA specifies the amount of a resource a user is entitled to utilize. In many cloud settings, providers would like to operate resources at high utilization while simultaneously respecting individual SLAs. There is typically a trade-off between these two objectives; for example, utilization can be increased by shifting away resources from idle users to "scavenger" workload, but with the risk of the former then becoming active again. We study this fundamental tradeoff by formulating a resource allocation model that captures basic properties of cloud computing systems, including SLAs, highly limited feedback about the state of the system, and variable and unpredictable input sequences. Our main result is a simple and practical algorithm that achieves near-optimal performance on the above two objectives. First, we guarantee nearly optimal utilization of the resource even if compared with the omniscient offline dynamic optimum. Second, we simultaneously satisfy all individual SLAs up to a small error. The main algorithmic tool is a multiplicative weight update algorithm and a primal-dual argument to obtain its guarantees. We also provide numerical validation on real data to demonstrate the performance of our algorithm in practical applications.

Funding: The authors' work was partially supported by the U.S. National Science Foundation [Grants CMMI 1552479, AF 1910423, and AF 1717947].

Keywords: cloud computing • online algorithms • multiplicative weights

1. Introduction

Cloud computing has motivated renewed interest in resource allocation, manifested in new consumption models (e.g., AWS spot pricing), as well as the design of resource-sharing platforms (Hindman et al. 2011, Vavilapalli et al. 2013). These platforms need to support a heterogenous set of users, also called tenants that share the same physical computing resource, e.g., CPU, memory, I/O bandwidth. Providers such as Amazon, Microsoft, and Google offer cloud services with the goal of benefiting from economies of scale. However, the inefficient use of resources — overprovisioning on the one hand or congestion on the other — could result in a low return on investment or in loss of customer goodwill, respectively. Hence, resource allocation algorithms are key for efficiently utilizing cloud resources.

To ensure quality of service, cloud offerings often come with a *service level agreement* (SLA) between the provider and the users. A SLA specifies the amount of a resource the user is entitled to consume. Perhaps the most common example is renting a virtual machine (VM) that guarantees an explicit amount of CPU, memory, etc. Naturally, VMs that guarantee more resources are more expensive. In this context, a simple allocation policy is to assign each user the resources specified by their SLAs. However, such an allocation can be wasteful, as users may not need the resource at all times. In principle, a dynamic allocation of resources can increase the total efficiency of the system. However, allocating resources dynamically without carefully accounting for SLAs can lead to user dissatisfaction.

Recent scheduling proposals address these challenges through work-maximizing yet fair schedulers (Zaharia et al. 2010, Ghodsi et al. 2011). However, such schedulers do not have explicit SLA guarantees. On the other hand, other works focus on enforcing SLAs (Curino et al. 2014, Grandl et al. 2016, Jyothi et al. 2016) but do not explicitly optimize the use of extra resources.

Our goal in this work is to understand the fundamental trade-off between high utilization of resources and SLA satisfaction of individual users. In particular, we design algorithms that guarantee both near-optimal utilization and the satisfaction of individual SLAs simultaneously. To that end, we formulate a basic model for online dynamic resource allocation. We focus on a single divisible resource, such as CPU or I/O bandwidth, that has to be shared among multiple users. Each user also has a SLA that specifies the fraction of the resource it expects to obtain. The actual demand of the user is in general time-varying and may exceed the fraction specified in the SLA. As in many real systems, the demand is not known in advance but rather arrives in an online manner. Arriving demand is either processed or queued up, depending on the resource availability. In many real-world scenarios, it is difficult to measure the actual demand size (see, e.g., Narasayya et al. (2013)). Accordingly, we assume that the system (and the underlying algorithm) receives only a simple *binary feedback* per user at any given time: whether the user queue is empty (the user's work arriving so far has been completed) or not. This is a plausible assumption in many systems, because one can observe workload activity, yet anticipating how much of the resource a job will require is more difficult. Additionally, it also models settings where demands are not known in advance.

Whereas online dynamic resource allocation problems have been studied in different contexts and communities (see Section 1.3 for an overview), our work aims to address the novel aspects arising in the cloud computing paradigm, particularly the presence of SLAs, the highly limited feedback about the state of the system, and a desired robustness over arbitrary input sequences. For the algorithm design itself, we pay close attention to practicality; our approach involves fairly simple computations that can be implemented with minimal overhead of space or time. Our algorithm achieves nearly optimal utilization of the resource as well as approximately satisfying the SLA of each individual user. We see two main use cases for the algorithm:

- In enterprise settings ("private cloud"), different applications or organizations share the same infrastructure. These often have SLAs, but providers would still like to maximize the return on investment (ROI) by maximizing utilization (Rasley et al. 2016).
- In public clouds, users buy VMs, which are offered at different "sizes" (which is practically the SLA). In addition, the service providers offer "best-effort" alternatives, such as Azure Batch (MS) or Spot instances (AWS). In our model, these services can be modeled by giving a SLA of zero. Here, satisfying the VM SLAs and achieving high utilization are both important; indeed, the provider is paid for the best-effort workloads only if it completes these jobs. Our work can be viewed as a principled way to accommodate such services and even give VMs better service than expected, an important consideration as public cloud offerings gradually become commoditized.

1.1. The Model

We consider the problem of having multiple tenants or users sharing a single resource, such as CPU, I/O, or networking bandwidth. For simplicity, we assume that the total resource capacity is normalized to 1. We have N users sharing the resource, a finite but possibly unknown discrete time horizon indexed $t = 1, \dots, T$, and an underlying queuing system. For each user i, we are also given an expected share of resource $\beta(i) \geq 0$ satisfying $\sum_{i=1}^{N} \beta(i) \leq 1$. The input is an online sequence of workloads $L_1, \dots, L_T \in \mathbf{R}_+^N$, where $L_t(i) \geq 0$ corresponds to i's workload arising at time t. The system maintains a queue $Q_t(i)$, denoting i's remaining work at time t. In our model, the decision maker does not have direct access to the values of the queues or the workloads. This allows us to consider settings where the job sizes are not known in advance and minimal information is available about the underlying system, a regular occurrence in many cloud settings. At time t, the following happens:

1. **Feedback:** The decision maker observes which queues are nonempty (the set of users i with $Q_t(i) > 0$, the active users), and which are empty ($Q_t(i) = 0$, the inactive users).
2. **Decision:** The decision maker updates user resource allocations $h_t(i)$, satisfying $\sum_i h_t(i) \leq 1$.
3. **Update:** The load $L_t(i)$ for each i arrives, and each user processes as much of the work from the queue plus the arriving workload as possible. The work completed by user i in step t is

$$w_t(i) := \min\{h_t(i), L_t(i) + Q_t(i)\}.$$

The queues at the end of the time step are updated accordingly:

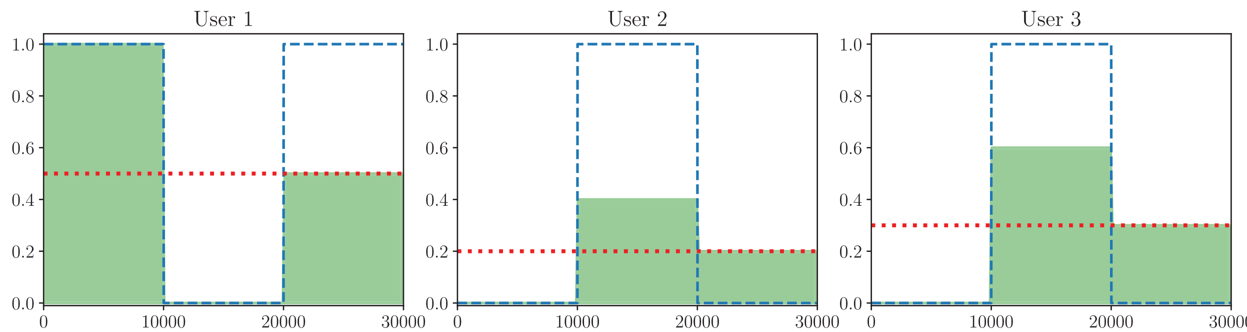
$$Q_{t+1}(i) = \max\{0, L_t(i) + Q_t(i) - h_t(i)\}.$$

We assess the performance of any algorithm based on two measures.

1. **Work Maximization:** The algorithm should maximize the total work completed over all users and thus utilize the resource as much as possible.
2. **SLA Satisfaction:** The algorithm should (approximately) satisfy the SLAs in the following manner. The work completed by user i up to any time $1 \leq t \leq T$ should be no less than the work completed for this user up to t if it were given a constant $\beta(i)$ fraction of the resource over the whole horizon.

Achieving either of the criteria on their own is straightforward. A greedy strategy that takes away resources from an idle user and gives them to any user whose queue is nonempty is approximately work-

Figure 1. (Color online) Example of Loads and Work for 3 Users



Notes. The dashed lines show each user's workload. The solid areas represent the work done by the users. The dotted lines depict SLAs.

maximizing (see Appendix C for details). On the other hand, to satisfy the SLAs, we give each user a static assignment of $h_t(i) := \beta(i)$ for all t. Naturally, the two criteria compete with each other; the following examples illustrate why these simple algorithms do not satisfy both simultaneously.

Example 1. We have a shared system with three users and corresponding SLAs $\beta(1) = 0.5, \beta(2) = 0.2$ and $\beta(3) = 0.3$. Loads are defined by

$$L_t(1) = \begin{cases} 1 & t = 1, \dots, T/3, 2T/3 + 1, \dots, T \\ 0 & t = T/3 + 1, \dots, 2T/3 \end{cases}, \text{ and } \quad L_t(2) = L_t(3) = 1 - L_t(1).$$

We assume that T is divisible by 3. In Figure 1, we show the three users' loads in blue dashed lines and the corresponding SLAs in dotted red lines. The static solution given by the SLAs, that is, $h_t(i) = \beta(i)$ for all t, ensures a total of $5T/6$ work done. However, the dynamic policy given by ensures T work is done (the green area). Moreover, it also ensures SLA satisfaction at all times. An alternative policy is

T	$[0, T/3]$	$[T/3 + 1, 2T/3]$	$[2T/3 + 1, T]$
$h_t(1)$	1	0	0.5
$h_t(2)$	0	0.4	0.2
$h_t(3)$	0	0.6	0.3

which is also work maximizing. However, it does not ensure SLA satisfaction. Indeed, this policy does not satisfy user 3's SLA at any time in $(T/3, 2T/3]$.

We remark that achieving both criteria is relatively simple if we allow the decision maker to observe demand or even the queue length. This can be achieved by first allocating to each user as much of the resource

t	$[0, T/3]$	$[T/3 + 1, 2T/3]$	$[2T/3 + 1, T]$
$h_t(1)$	1	0	0
$h_t(2)$	0	1	0
$h_t(3)$	0	0	1

as necessary up to their SLA and then distributing the remaining resource arbitrarily among users with additional demand. The versatility of our setting stems from the limited feedback in the form of binary information about idle and busy users. In our cloud computing context, full demand information or even visible queue lengths are unrealistic assumptions.

1.2. Our Results and Contributions

We design a simple and efficient online algorithm that achieves approximate work maximization as well as approximate SLA satisfaction even in the limited feedback model that we consider. For work maximization, we analyze the performance by comparing our algorithm to the *optimal offline dynamic allocation* that knows all the data up front. In contrast, our online algorithm receives limited feedback even in an online setting. Thus, our aim is to minimize the quantity

$$\text{work}_{\mathbf{h}_1^*, \dots, \mathbf{h}_T^*} - \text{work}_{\text{alg}} = \sum_{t=1}^{T} \sum_i w_t^*(i) - \sum_{t=1}^{T} \sum_i w_t(i),$$

where $\text{work}_{\mathbf{h}_1^*, \dots, \mathbf{h}_T^*}$ is the optimal offline work done by dynamic allocations, $\mathbf{h}_1^*, \dots, \mathbf{h}_T^*$, $\mathbf{w}_t^* = (w_t^*(1), \dots, w_t^*(N))$ is the work done at time t by these allocations, and work_{alg} is the work done by the algorithm with allocations $\mathbf{h}_1, \dots, \mathbf{h}_T$ and work $\mathbf{w}_t = (w_t(1), \dots, w_t(N))$ at time t. The objective of the decision maker is to minimize this quantity by constructing a sequence of good allocations that approach the best allocations in hindsight. Note that our benchmark is dynamic rather than the more common static offline optimum usually considered in regret minimization (Arora et al. 2012, Shalev-Shwartz 2012, Hazan 2019). Similarly, for SLA satisfaction, our benchmark is the total work done for a user if they were given $\beta(i)$ resources for each time $1 \le t \le T$. We give a bicriteria online algorithm that achieves nearly the same performance as the benchmarks if the resources for the latter are slightly more constrained than that of the algorithm. Algorithm 1, which we formally describe in Section 2, follows a multiplicative weight approach. The idea is to boost the allocations of active users by a factor greater than 1, with more emphasis on users with current allocation below their SLA. This intuition translates into a simple update that ensures high utilization of the resource and SLA satisfaction, formally, as follows.

Theorem 1. *For any input parameter $0 < \varepsilon \le 1/10$, SLAs $\beta = (\beta(1), \ldots, \beta(N))$ satisfying $\beta(i) \ge 2\varepsilon/N$ and online loads $L_1, \ldots, L_T \in \mathbf{R}^N_{\ge 0}$, Algorithm 1 achieves the following guarantees:*

1. **Approximate Work Maximization.** *Let $\mathbf{h}^*_1, \ldots, \mathbf{h}^*_T \in [0,1]^N$ be an optimal offline sequence of allocations such that $\sum_i h^*_t(i) = 1$ for all $1 \le t \le T$. Then*

$$\mathrm{work}_{\mathrm{alg}} \ge (1-\varepsilon)\mathrm{work}_{\mathbf{h}^*_1,\ldots,\mathbf{h}^*_T} - \mathcal{O}\Big(N\varepsilon^{-2}\log(N/\varepsilon)\Big).$$

2. **Approximate SLA Satisfaction.** *There exists $\tilde{p} = \tilde{p}(N,\varepsilon) = \mathcal{O}(N^2\varepsilon^{-3}\log(N/\varepsilon))$ such that for any user i and time t, if we take $\mathbf{h}'_1, \ldots, \mathbf{h}'_T \in [0,1]^N$ to be any sequence of allocations with $h'_t(i) \le \beta(i)$, then*

$$\sum_{\tau=1}^{t} w_\tau(i) \ge (1-2\varepsilon)\sum_{\tau=1}^{t} w'_\tau(i) - \beta(i)\tilde{p},$$

where $\mathbf{w}'_t$ is the work performed by the allocations $\mathbf{h}'_1, \ldots, \mathbf{h}'_T$.

The first part of Theorem 1 asserts that if T is known for the system, we can achieve $\mathrm{work}_{\mathrm{offline}} - \mathrm{work}_{\mathrm{alg}} \le \mathcal{O}(T^{2/3}\log T)$ *with $\varepsilon = \Theta(N^{1/3}/T^{1/3})$. However, this choice of ε is suboptimal for SLA satisfaction. We can achieve an improved bound for SLA satisfaction (when $t = T$) by picking $\varepsilon = \Theta(N^{1/3}/T^{1/4})$. If T is unknown, we can use a standard doubling trick; see, for example,* Shalev-Shwartz (2012). *Summarizing, we obtain the following result.*

Corollary 1. *For $\varepsilon = \Theta(N^{1/3}/T^{1/4})$, Algorithm 1 guarantees*

$$\mathrm{work}_{\mathbf{h}^*_1,\ldots,\mathbf{h}^*_T} - \mathrm{work}_{\mathrm{alg}} \le \mathcal{O}(N^{1/3}T^{3/4} + N^{1/3}T^{1/2}\log(NT)) = \mathcal{O}(N^{1/3}T^{3/4}),$$

where $\mathbf{h}^*_1, \ldots, \mathbf{h}^*_T$ are optimal offline dynamic allocations.

As T grows, Corollary 1 guarantees that the rate of work done by our algorithm $\mathrm{work}_{\mathrm{alg}}/T$ approaches the rate of work done by the optimal dynamic solution $\mathrm{work}_{\mathbf{h}^*_1,\ldots,\mathbf{h}^*_T}/T$. We emphasize again that this is a stronger guarantee using the much more powerful optimal dynamic solution as a benchmark, rather than the typical static allocation used in regret analysis for online algorithms. Such a guarantee can be obtained in our model because the incomplete work remains stored in the queues until we are able to finish it; this allows the algorithm to catch up with the incomplete work.

The second result in Theorem 1 states that the work done by any individual user is comparable to the work done by their promised SLA. In other words, the user's queue length is not much larger than it would be under a static SLA allocation. By using $\varepsilon = \Theta(N^{1/3}/T^{1/4})$, we obtain the following result.

Corollary 2. *Let $\varepsilon = \Theta(N^{1/3}/T^{1/4})$. For a user i, $Q_T(i) \le Q'_T(i) + \mathcal{O}(NT^{3/4}\log(NT))$, where Q_t is the queue given by Algorithm 1 and Q'_t is the queue induced by any dynamic policy $\mathbf{h}'_1, \ldots, \mathbf{h}'_T$ with $h'_t(i) \le \beta(i)$.*

We remark that we need to bound the SLAs away from ε/N; in Theorem 1, we use the bound $\beta(i) \ge 2\varepsilon/N$. This condition is necessary in our analysis to guarantee that there is an overprovisioned ($h_t(i) > \beta(i)$) user from which we can move allocation to an underprovisioned user. See Theorem 4 for details and a more relaxed bound.

Corollary 1's guarantee is near-optimal asymptotically in T in terms of work maximization, as the following result shows. The proof of this result appears in Appendix D.

Theorem 2. *For any online deterministic algorithm $\mathcal{A}$ for our model, there is a sequence of online loads $L_1, \ldots, L_T$ such that $\mathrm{work}_{\mathbf{h}^*_1,\ldots,\mathbf{h}^*_T} - \mathrm{work}_{\mathcal{A}} = \Omega(\sqrt{T})$, where $\mathbf{h}^*_1, \ldots, \mathbf{h}^*_T$ are optimal offline dynamic allocations.*

Our algorithm follows a mirror descent approach (Ben-Tal and Nemirovski 2001, Hazan 2019). Unable to see the lengths of the queues, a (naive) approach is to pretend that active users have gigantic queues. From this approach, we extract a simple update rule that multiplicatively boosts active users; however, active users who are under their SLA are boosted slightly more than other active users. If inactive users are assigned more than ε of the resource, where ε is the algorithm parameter, active users ramp up their allocation in few iterations. In the opposite case, at least $1-\varepsilon$ of the resource is assigned to active users, and the slight boost to users below their SLA ensures a healthy rebalancing of the resource. We show that this efficient heuristic strategy is enough to achieve approximate work maximization and SLA satisfaction. We remark that the mirror descent interpretation is used only to provide intuition for the algorithm, and our proofs follow a different path than the usual mirror descent analysis. Later, we detail a slight modification that enjoys the same theoretical guarantees as Algorithm 1 but in practice exhibits more desirable behavior (see Algorithm 2). Intuitively, among active users, the modified algorithm tries to keep allocations proportional to their SLAs. This behavior is appealing;

for example, a user A with twice the SLA of another user B would expect in practice to perform at least twice as much work. Similarly, user B would expect to receive no less than half of user A's allocation. This second algorithm exhibits another interesting feature; it can be applied to overcommitted regimes with $\sum_{i=1}^{N}\beta(i) > 1$, remain work-maximizing, and satisfy a normalized version of SLA satisfaction (see Section 3.4).

The analysis of the algorithm relies on a primal-dual fitting approach. For work maximization, we can write the offline dynamic optimal allocation as a solution to a linear program and then construct feasible dual solutions with objective value close to the algorithm's resource utilization. A crucial ingredient of the algorithm is the use of entropic projection on the truncated simplex, which ensures that every user gets at least a ε/N fraction of the resource at all times. Intuitively, this means that any user with a nonempty queue will recover their SLA requirement in a few steps.

We do an extensive analysis of our algorithm on synthetic data as well as real data obtained from CPU traces of a production service in Microsoft's cloud. We aim to quantify the performance of the algorithm on three objectives: (i) work maximization, (ii) SLA guarantee, and (iii) queue behavior. Although our theoretical results give guarantees for these objectives, we show experimentally that the algorithm exceeds these guarantees. We benchmark the algorithm against natural online algorithms, such as a static allocation as given by the SLA guarantee, or the algorithm that aims to proportionally distribute the resource among active clients (according to their SLA). We also benchmark against offline algorithms that know all input data up front; our algorithm's performance on various measures is comparable to the offline algorithms.

This work is organized as follows. In Section 2, we present the preliminaries and the basic version of the multiplicative weight algorithm. Section 3 contains the proof of Theorem 1 in the bicriteria form. We split the proof into two parts: work maximization in Section 3.2 and SLA satisfaction in 3.3. In Section 3.4, we present the extension of our algorithm and its guarantees. Finally, in Section 4, we present numerical experiments that empirically validate our results, and we conclude in Section 5.

1.3. Related Work

There has been growing interest in resource allocation problems arising from cloud computing applications both from a practical as well as a theoretical standpoint (Hindman et al. 2011, Narasayya et al. 2013, Vavilapalli et al. 2013, Curino et al. 2014, Menache and Singh 2015, Narasayya et al. 2015, Grandl et al. 2016, Jyothi et al. 2016, Rasley et al. 2016). The focus of many of these works has been to understand the trade-offs between efficiency and ensuring guarantees to individual users.

On the more theoretical side, the cloud computing allocation problem has been modeled as a stochastic allocation problem (Maguluri et al. 2012, 2014; Maguluri and Srikant 2014). The underlying models draw inspiration from a large body of work on stochastic network control, originating from the seminal work of Tassiulas and Ephremides (1990, 1993), followed by additional related research, for example, on uplink and downstream scheduling in wireless networks (Neely 2007, 2008). The analytical results in these papers characterize the stability region of the arrival processes under certain stochastic assumptions (e.g., i.i.d. processes), and suggest algorithms that achieve maximal throughput. The main distinction between these works and ours is that we assume an adversarial input; that is, we do not make stationary distributional assumptions on the input. Another difference is that our model centers on the notion of an SLA that is known to the algorithm. This allows us to address the overcommited case (see Algorithm 2), which is especially relevant in cloud settings.

Despite these modeling differences, there are some parallels worth mentioning. For example, we design algorithms with rate of work $\mathrm{work}_{\mathrm{alg}}/T$ approaching the optimal offline rate of work; this translates to the average delay $\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}Q_t(i)$ converging to the optimal offline average delay. This result can be compared with the limiting behavior of the Markov process in many of the stochastic network models. For example, Neely (2008) studied a system of N users connecting to a server via ON/OFF channels. The paper presents an algorithm that ensures bounded average delay $\limsup_T \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T}\sum_{i=1}^{N}Q_i(t)\right]$ for any input within the interior of the stability region; here, $Q_i(t)$ is the i-th user's backlog (queue length) at time t. Much of the stochastic network literature assumes full knowledge of queue lengths, for example, the LCQ policy in Tassiulas and Ephremides (1993), although there are studies that limit the information available to the decision maker in a similar fashion to our model (see Neely 2007, Li and Neely 2009, Shirani-Mehr et al. 2010, and Maguluri and Srikant 2014).

More broadly, the general problem of online resource allocation has been studied in both stochastic and adversarial settings; we refer the reader to the books (Albers 2003, Borodin and El-Yaniv 2005, Srikant and Ying 2013) on the topic. Our work differs from the aforementioned lines of research by combining the three following elements already present in the literature. First, as mentioned above, we digress from the stochastic arrival model to the adversarial setting and worst-case analysis. This makes our algorithm robust against unpredictable users' demands. For instance, demands in the morning could be totally different from demands in the afternoon or the morning of the next day. We are able to provide a single strategy that adapts easily to any scenario. Second, we provide simple online algorithms that perform well even under limited feedback, a typical situation in cloud systems in which we can determine the utilization of a resource only after it has been allocated. Finally, we consider SLA satisfaction as a measure of user contentment and seek to satisfy it up to a small error.

There is now an extensive literature devoted to the pricing of cloud computing services. In (Macías and Guitart 2011) the authors study a genetic model for generating a suitable pricing function in the cloud market. In (Gera and Xia 2011), pricing is studied via a revenue management formulation to address resource provisioning decisions. See also Passacantando et al. (2016) and Sharma et al. (2012) for more pricing models. A closely related topic is fairness in resource allocation (Zaharia et al. 2010, Ghodsi et al. 2011). Although we do not directly consider pricing, work maximization could be interpreted as a way to obtain extra revenue by allocating unused resources to active users.

More recently, there has been work considering overcommitment in the cloud (Gordon et al. 2011, Dabbagh et al. 2015, Cohen et al. 2019), that is, selling resources beyond server capacity. One of the objectives of overcommitment is to reduce the number of servers opened in order to minimize energy consumption. In our basic model, we do not assume overcommitment, yet our algorithm can still be applied to that setting (see Algorithm 2). Specifically, we obtain a normalized version of SLA satisfaction in the overcommitment setting, where the guarantees depend on how much the system is overcommitted (see Section 3.4).

A fundamental tool in our design is *mirror descent* algorithms (Ben-Tal and Nemirovski 2001). These first-order iterative algorithms have been widely used in optimization (Ben-Tal and Nemirovski 2001), online optimization, and machine learning (Shalev-Shwartz 2012, Mohri et al. 2018, Hazan 2019) to generate update policies under limited feedback. Similarly, multiplicative weight algorithms have been widely studied in optimization (Plotkin et al. 1995, Arora et al. 2012), online convex optimization (Hazan 2019), online competitive analysis (Buchbinder and Naor 2009), and learning theory (Freund and Schapire 1997, Shalev-Shwartz 2012). Our results bear some resemblance to regret analysis, where typically the benchmark is the optimal offline static policy (Freund and Schapire 1997, Abernethy et al. 2008, Bubeck et al. 2012, Shalev-Shwartz 2012, Hazan 2019); the use of a dynamic benchmark (as in our work) is scarcer in the literature, see, for example, Zinkevich (2003), Hall and Willett (2015), Mokhtari et al. (2016), and Zhang et al. (2017).

2. Algorithm

2.1. Preliminaries

For $N \geq 1$, we identify the set of users with the set $[N] = \{1, \ldots, N\}$. For $0 < \varepsilon < 1$, we call an allocation $\mathbf{h} = (h(1), \ldots, h(N)) \in [0,1]^N$ a $(1-\varepsilon)$-allocation if $\sum_i h(i) \leq 1 - \varepsilon$. We assume $N \geq 2$, that is, the system consists of at least two users.

For any t, we define the set of active users at that time as the set of users with nonempty queue and denote this set by A_t. Observe that $h_t(i) = w_t(i)$ for all active users. Let $B_t = [N] \backslash A_t$ be the sets of users with empty queues at time t; we call these users inactive. A_t and B_t correspond to the feedback given to the decision maker. Also, let $A_t^1 = \{i \in A_t : h_t(i) < \beta(i)\}$ be the set of active users with allocation below their SLA and $A_t^2 = A_t \backslash A_t^1$ be the set of active users receiving at least their SLAs.

We assume without loss of generality that the allocations set by the decision maker always add up to 1. We propose an algorithm that uses a multiplicative weight strategy to boost a subset of users by multiplying their allocation by a factor greater than 1. Because the allocations do not sum to 1 after applying the update, we then project them onto the simplex using the KL-divergence metric. Furthermore, to ensure that no user gets an allocation arbitrarily close to zero, we in fact project onto the *truncated simplex*,

$$\Delta_\varepsilon = \{\mathbf{x} = (x(1), \ldots, x(N)) : \|\mathbf{x}\|_1 = 1, x(i) \geq \varepsilon/N, \ \forall i\}.$$

To fix the notation, let $\pi_{\Delta_\varepsilon}(\cdot)$ be the *projection function* onto Δ_ε using Kullback-Leibler divergence (KL-divergence for short), that is, $\pi_{\Delta_\varepsilon}(\mathbf{y}) := \operatorname{argmin}_{\mathbf{x} \in \Delta_\varepsilon} \sum_i x(i) \log(x(i)/y(i))$, where $\mathbf{y} = (y(1), \ldots, y(N)) \in \mathbf{R}^N_{\geq 0}$. In Appendix A, we show how to efficiently compute this projection. The following proposition states some basic facts that are useful in our analysis. The proof appears in Appendix A.

Proposition 1. *Let* $\mathbf{y} \in \mathbf{R}^N_+, \mathbf{x} = \pi_{\Delta_\varepsilon}(\mathbf{y})$ *and* $S = \{i : x(i) = \varepsilon/N\}$. *Then:*

(a) *If* $y(1) \leq y(2) \leq \cdots \leq y(N)$, *then* $S = \{1, \ldots, k\}$ *for some* $k \geq 0$.

(b) $x(i) = y(i)e^{\mu_i}C$, *where* $C = \left(1 - \varepsilon N|S| / \sum_{j \in S} y(j)\right)$, $\mu_i \geq 0$ *for all i and* $\mu_i = 0$ for $i \notin S$.

(c) $\mathbf{x}$ *can be computed in* $\mathcal{O}(N \log N)$ *time.*

2.2. The Multiplicative Weight Algorithm

We propose an algorithm that follows a multiplicative weight strategy (see Algorithm 1). We describe here the basic approach given by the mirror descent algorithm. In Section 3.4, we present an extension of the algorithm that in practice shows a better relation between the allocations and the ratios between the SLAs.

Algorithm 1 (Multiplicative Weight Update Algorithm).
Input: Parameters $0 < \varepsilon \le \frac{1}{10}, 0 < \eta < \frac{1}{3}$.
1. Initialization: $\mathbf{h}_1$ any allocation over Δ_ε and $\lambda = \frac{\varepsilon^2}{8N}$.
2. **For** $t = 1, \dots, T$ **do**
3. Set allocation $\mathbf{h}_t$.
4. Read active and inactive users A_t and B_t. $A_t^1 = \{i \in A_t : h_t(i) < \beta(i)\}$, $A_t^2 = A_t \backslash A_t^1$.
5. Set gain function $g_t(i) = \begin{cases} 1+\lambda & i \in A_t^1 \\ 1 & i \in A_t^2 \\ 0 & i \in B_t \end{cases}$.
6. Update allocation:
$$\hat{h}_{t+1}(i) = h_t(i) e^{\eta g_t(i)}.$$
$$\mathbf{h}_{t+1} = \pi_{\Delta_\varepsilon}(\hat{h}_{t+1}).$$

Intuitively, the algorithm boosts active users at the expense of inactive ones and boosts users slightly more if they are currently under their SLA. The algorithm update rule comes from a mirror descent approach applied to a Lagrangian relaxation of a work-maximization linear function at time t. More formally, under the assumption that active users have a huge queue, we aim to maximize the objective $\sum_{i \in A_t} w_t(i)$ subject to $w_t(i) \ge \beta(i)$ for $i \in A_t$. The update rule is obtained after applying a mirror descent with a KL-divergence distance generating function over the simplex to the Lagrangian relaxation of the previous problem (see Ben-Tal and Nemirovski 2001, Boyd and Vandenberghe 2004). We restrict the projection to the truncated simplex so that no user gets an allocation too close to 0. We use this update rule solely to guide the algorithm's decisions; however, the proofs of work maximization and SLA satisfaction do not follow from the standard analysis of mirror descent.

3. Analysis

To give the analysis of the algorithm and prove Theorem 1, we prove the following stronger guarantees about Algorithm 1. We compare its performance to the optimal offline dynamic strategy that uses at most a $1 - 4\varepsilon$ fraction of the resources at each time step.

Theorem 3. *Given loads $L_1, \dots, L_T$, for any $\varepsilon > 0$ and $\eta > 0$ such that $\varepsilon \le 1/10$, Algorithm 1 guarantees*

$$\text{work}_{\mathbf{h}_1^*, \dots, \mathbf{h}_t^*} - \text{work}_{alg,t} \le 8 \frac{N}{\varepsilon^2 \eta} \ln(N/\varepsilon),$$

for any time $1 \le t \le T$, where $\mathbf{h}_1^*, \dots, \mathbf{h}_T^*$ is the optimal offline sequence of $(1-4\varepsilon)$-allocations and $\text{work}_{alg,t} = \sum_i \sum_{\tau=1}^t w_\tau(i)$ is the work done by Algorithm 1 until time t.

The first guarantee of Theorem 1 regarding work maximization now follows simply from Theorem 3. Given any offline dynamic policy $\mathbf{h}_1, \dots, \mathbf{h}_T$ such that $\sum_i h_t(i) = 1$, we define $\overline{\mathbf{h}}_t := (1-4\varepsilon)\mathbf{h}_t$, which satisfies the assumption of Theorem 3. Now we have

$$\begin{aligned} \text{work}_{\text{alg}} &\ge \text{work}_{\overline{\mathbf{h}}_1, \dots, \overline{\mathbf{h}}_T} - 8 \frac{N}{(\varepsilon/4)^2 \eta} \ln(4N/\varepsilon) \\ &\ge (1-4\varepsilon) \cdot \text{work}_{\mathbf{h}_1, \dots, \mathbf{h}_T} - 2000 \frac{N}{\varepsilon^2 \eta} \ln(N/\varepsilon), \end{aligned}$$

where the first inequality follows from Theorem 3. To argue the second, let $\mathbf{w}_1, \dots, \mathbf{w}_T$ and $\overline{\mathbf{w}}_1, \dots, \overline{\mathbf{w}}_T$, respectively, denote the work performed by allocations $\mathbf{h}$ and $\overline{\mathbf{h}}$. Then, $(1-4\varepsilon)\mathbf{w}_1, \dots, (1-4\varepsilon)\mathbf{w}_T$ are feasible work patterns that the allocations $\overline{\mathbf{h}}_1, \dots, \overline{\mathbf{h}}_T$ could do, since in this setting we have $1-4\varepsilon$ capacity and the same workload. Therefore, $(1-\varepsilon)\text{work}_{\mathbf{h}_1, \dots, \mathbf{h}_T} \le \text{work}_{\overline{\mathbf{h}}_1, \dots, \overline{\mathbf{h}}_T}$, because the users try to use their allocations at maximum.

Similarly, for SLA satisfaction, we prove a stronger bicriteria result that implies the SLA guarantee in Theorem 1.

Theorem 4. *Let $0 < \varepsilon \le 1/10$ and $0 < \eta \le 1/3$. Take any SLAs $\beta(1), \dots, \beta(N)$ such that $\beta(i) \ge e^{\eta(1+\lambda)}\varepsilon/N$, where $\lambda = \varepsilon^2/8N$, and let $\tilde{p} = 32N^2/\varepsilon^3\eta \ln(N/\varepsilon)$. Then, for any user i and time $t \le T - \tilde{p}$, if we take $\mathbf{h}_1', \dots, \mathbf{h}_T'$ to be allocations such that $h_t'(i) = (1-2\varepsilon)\beta(i)$, the work done by Algorithm 1 for user i satisfies*

$$\sum_{\tau=1}^{t+\tilde{p}} w_\tau(i) \ge \sum_{\tau=1}^{t} w_\tau'(i),$$

where $\mathbf{w}_t'$ is the work done by the allocations $\mathbf{h}_1', \dots, \mathbf{h}_T'$. Moreover, $\sum_{\tau=1}^t w_\tau(i) \ge \sum_{\tau=1}^t w_\tau'(i) - \beta(i)\tilde{p}$.

3.1. The Offline Formulation

Before presenting the proof of Theorem 3, we state the offline LP formulation of the maximum work problem for $(1-\varepsilon)$-allocations. We denote by $\mathbf{w}_t = (w_t(1), \dots, w_t(N))$ the work done for each user at time t. Given loads $L_1, \dots, L_T$, the offline formulation and its dual LP are given in Figure 2. As written, the dual LP includes a change of variable; see Appendix B for details. Constraints (2) state that the work done for any user up to time t by the allocation cannot exceed the user's loads up to that time. Constraints (3) limit the work performed at time t to at most a $1-\varepsilon$ fraction of the resource. The LP (D_ε) will be of special importance in the analysis. Using our algorithm, we will construct a dual feasible solution.

Observe that (P_ε) is feasible and bounded since the feasible region is a nonempty polytope. Let v_{P_ε} be the optimal value of (P_ε). The following proposition gives a simple characterization of v_{P_ε}; the proof appears in Appendix B.

Proposition 2. $v_{P_\varepsilon} = \min_{0 \le t \le T} \left(\sum_{s=1}^t \sum_i L_s(i) + (1-\varepsilon)(T-t) \right)$.

Figure 2. The Primal and Dual LP Formulation for the Maximum Work Problem

$$(P_\varepsilon)\quad \max \quad \sum_{i=1}^{N}\sum_{t=1}^{T} w_t(i)$$

s.t.

$$\forall t,i \quad \textstyle\sum_{s=1}^{t} w_s(i) \le \sum_{s=1}^{t} L_s(i) \qquad (1)$$

$$\forall t \quad \textstyle\sum_{i=1}^{N} w_t(i) \le 1-\varepsilon \qquad (2)$$

$$\forall t \quad \mathbf{w}_t \ge 0 \qquad (3)$$

$$(D_\varepsilon)\quad \min \quad \sum_{i=1}^{N}\sum_{t=1}^{T} L_t(i)\gamma_t(i) + (1-\varepsilon)\sum_{t=1}^{T}\beta_t$$

s.t.

$$\forall t,i \quad \gamma_t(i)+\beta_t \ge 1 \qquad (4)$$

$$\forall t \quad \gamma_t \ge \gamma_{t+1} \qquad (5)$$

$$\forall t \quad \beta_t,\gamma_t \ge 0 \qquad (6)$$

3.2. Work Maximization

In this section, we prove Theorem 3. Our first lemma characterizes the implications of the update rule. The proof follows from a careful analysis of the dynamics using the KL-divergence and appears in Appendix B.

The first result of the lemma shows the behavior of active users' allocations when the system is underutilized ($\le 1-\varepsilon$). In this case, all of the active users receive a multiplicative boost in their allocation. The second result shows a more general behavior (see also Lemma 2). In this case, active users with allocation below their SLA do not decrease their allocations, whereas the other active users might decrease their allocation, but in this case, the multiplicative penalization will be less severe.

Lemma 1. *Let $c = \varepsilon\eta/(4N)$. Then, Algorithm 1 satisfies the following:*

1. *Suppose $\sum_{i\in A_t} h_t(i) \le 1-\varepsilon$.* If $i \in A_t$, *then* $h_{t+1}(i) \ge h_t(i)(1+c)$.
2. *In general, Algorithm 1 satisfies $h_{t+1}(i) \ge h_t(i)$ for $i \in A_t^1$ and $h_{t+1}(i) \ge h_t(i)(1-\varepsilon c)$* for $i \in A_t^2$.

Proof of Theorem 3. Given loads $L_1,\dots,L_T \in \mathbf{R}_+^N$, consider the following $\{0,1\}$-matrix M of dimension $N \times T$ that encodes the information about the status of queues obtained while running Algorithm 1:

$$M_{i,t} = \begin{cases} 0 & i\text{'s queue is empty at } t, Q_t(i)=0,\\ 1 & i\text{'s queue is not empty at } t, Q_t(i)>0.\end{cases}$$

Let $\tilde{s} = \ln(N/\varepsilon)/(\varepsilon c)$, where c is defined in Lemma 1. Now, pick $s^\star$ to be the maximum nonnegative integer s (including 0) such that

$$\sum_{t=1}^{s}\sum_i L_t(i) \le \sum_{t=1}^{s+\tilde{s}}\sum_i w_t(i) \qquad (7)$$

Claim 1. *Consider any block of time $[r, r+\tilde{s}]$ where $r > s^\star$; then there exists a user i such that $M_{i,r'} = 1$ for all $r' \in [r, r+\tilde{s}]$.*

Proof. Suppose not. Then we claim that $s = r$ satisfies condition (1). Consider any user i and let $r_i' \in [r, r+\tilde{s}]$ be such that $M_{i,r_i'} = 0$. Then, work done by the user i up to time $r+\tilde{s}$ is at least

$$\sum_{t=1}^{r+\tilde{s}} w_t(i) \ge \sum_{t=1}^{r_i'} w_t(i) = \sum_{t=1}^{r_i'} L_t(i) \ge \sum_{t=1}^{r} L_t(i).$$

Now summing over all i, we get the desired contradiction. □

We now prove the following claim that shows that the algorithm ensures that, on average, the total resource utilization after $s^\star$ is close to $1-4\varepsilon$. The proof of the claim relies on Lemma 1 and appears in Appendix B.

Claim 2. *Let $B = [r, r+\tilde{s})$ with $r > s^\star$ be a consecutive block of $\tilde{s}$ time steps, and let $B' = \{t \in B : \sum_{j\in A_t} h_t(j) \le 1-\varepsilon\}$ be the time steps in B with low utilization. Then, $|B'| \le 4\varepsilon\tilde{s}$, and therefore, $\sum_{t=s^\star+1}^{T}\sum_i w_t(i) \ge (1-4\varepsilon)(T-s^\star) - \tilde{s}$.*

Now, consider the following feasible dual solution of $(D_{4\varepsilon})$; $\gamma_t(i) = 1, \beta_t = 0$ for all users i and $t = 1,\dots,s^\star$, and $\gamma_t(i) = 0, \beta_t = 1$ for all users i and $t = s^\star+1,\dots,T$. Observe that $\sum_{t=1}^{T}\beta_t = T - s^\star$. For optimal $(1-4\varepsilon)$-allocations $\mathbf{h}_1^*\dots,\mathbf{h}_T^*$, we obtain

$$\begin{aligned}
\text{work}_{\mathbf{h}_1^*,\dots,\mathbf{h}_T^*} &\le v_{\text{dual}}(\gamma_1,\dots,\gamma_T,\beta_1,\dots\beta_T) && \text{(work duality)}\\
&= \sum_{t=1}^{s^\star}\sum_i L_t(i) + (1-4\varepsilon)(T-s^\star)\\
&\le \sum_{t=1}^{s^\star+\tilde{s}}\sum_i w_t(i) + (1-4\varepsilon)(T-s^\star) && \text{(choice of } s^\star)\\
&\le \sum_{t=1}^{s^\star}\sum_i w_t(i) + \tilde{s} + \sum_{t=s^\star+1}^{T}\sum_i w_t(i) + \tilde{s} && \text{(Claim 2)}\\
&= \text{work}_{\text{alg}} + 8\frac{N}{\varepsilon^2\eta}\ln(N/\varepsilon).
\end{aligned}$$

where we have used $\sum_{t=s^\star+1}^{s^\star+\tilde{s}} w_t(i) \le \tilde{s}$ and the definition of $\tilde{s}$. □

3.3. SLA Satisfaction

In this section, we prove Theorem 4. Recall that $\lambda = \varepsilon^2/(8N)$ and $A_t^1 = \{i \in A_t : h_t(i) < \beta(i)\}$ are the set of active users receiving less than their SLAs and that $A_t^2 = A_t \backslash A_t^1$ is the set of active users receiving at least their SLA. Analogous to Lemma 1, we have the following lemma, whose proof appears in Appendix B.

Lemma 2. *Assume that $\varepsilon \le 1/10, \eta \le 1/3$ and $\beta(i) \ge 2\varepsilon/N$ for all users. Then, for any $i \in A_t^1$, Algorithm 1 guarantees $h_{t+1}(i) \ge h_t(i)(1 + c')$, where $c' = \varepsilon\eta\lambda/(2N)$.*

Proof of Theorem 4. Let $\tilde{p} = \lceil \ln(N/\varepsilon)/\ln(1 + c') \rceil$, where c' is defined in Lemma 2. Now, we proceed by induction on t to prove that $\sum_{\tau=1}^{t+\tilde{p}} w_\tau(i) \ge \sum_{\tau=1}^{t} w'_\tau(i)$, where $\mathbf{w}'_t$ is the work done by the allocations $\mathbf{h}'_1, \dots, \mathbf{h}'_T$. Clearly, the case $t = 0$ is direct.

Take $t \ge 1$ and suppose that the result is true for $t - 1$. If there exists $r \in [t, t + \tilde{p}]$ such that user i's queue is empty, then

$$\sum_{\tau=1}^{t+\tilde{p}} w_\tau(i) \ge \sum_{\tau=1}^{r} w_\tau(i) = \sum_{\tau=1}^{r} L_\tau(i) \ge \sum_{\tau=1}^{t} w'_\tau(i).$$

Therefore, assume that for all $\tau \in [t, t + \tilde{p}]$ we have that user i's queue is nonempty. By the induction hypothesis,

$$\sum_{\tau=1}^{t-1+\tilde{p}} w_t(i) \ge \sum_{\tau=1}^{t-1} w'_\tau(i).$$

In order to complete the proof, we need to prove that $w_{t+\tilde{p}}(i) \ge w_t{}'(i)$. We proceed as follows. Suppose that for all $\tau \ge t$ we have $w_\tau(i) < (1 - \varepsilon)\beta(i)$. By Lemma 2, at each time $\tau \in [t, t + \tilde{p}]$ the allocation of user i increases multiplicatively by a rate $(1 + c')$. Therefore,

$$w_{t+\tilde{p}}(i) \ge \frac{\varepsilon}{N}(1 + c')^{\tilde{p}} \ge 1 \ge \beta(i),$$

a contradiction. From the previous analysis, we obtain the existence of $\tau^\star \in [t, t + \tilde{p}]$ such that $w_{\tau^\star}(i) \ge (1 - \varepsilon)\beta(i)$. By using Lemmas 1 and 2, we can show that the allocation $h_\tau(i)$ will never go below $(1 - \varepsilon c)(1 - \varepsilon)\beta(i)$ for all $\tau \ge \tau^\star$. In particular, $w_{t+\tilde{p}}(i) \ge (1 - \varepsilon c)(1 - \varepsilon)\beta(i) \ge (1 - 2\varepsilon)\beta(i) \ge w_t{}'(i)$. □

3.4. Extension to Proportionality and Overcommitment

In the previous subsections, we have introduced the first version of the multiplicative weight algorithm. We explained how we deduced our algorithm using mirror descent and proved its theoretical guarantees. Even though Algorithm 1 guarantees individual SLA satisfaction, this simple policy can lead to undesirable results that do not respect the ratio between allocations. If one user has an SLA twice the size of another, it would be reasonable for the former to expect allocations at least twice as big as the latter's if both are consistently busy. Likewise, the second user would expect allocations no less than half of the first user's. In other words, both users should expect shares that respect the ratio between their SLAs.

To illustrate this unsatisfactory behavior in Algorithm 1, we run it with three users having SLAs $\beta(1) = 0.5, \beta(2) = 0.3$ and $\beta(3) = 0.2$. We set $\eta \le 1/3$ and $\varepsilon \le 1/10$. For simplicity, the initial allocation will be uniform. In our example, user 1 is always idle. User 2 consistently demands one unit of resource. User 3 begins idle and remains so until user 2's allocation reaches $1 - \varepsilon$. This takes roughly $\eta\varepsilon^{-1}$ time steps; call this time t_0. Starting at time t_0, user 3 demands unit loads every time step for the rest of the horizon. Initially, the allocations are uniformly $1/3$ for everyone. Between time 1 and t_0, user 2's allocation increases until it hits $1 - \varepsilon$, since this user is the only active user. After t_0, user 3 becomes active and has an allocation below his or her SLA. Therefore, the algorithm redistributes allocation from user 2 to 3 until user 3's allocation hits 0.2. After this, allocations remain stable at approximately $h_t(1) = \frac{\varepsilon}{3}$, $h_t(2) = 0.8 - \frac{\varepsilon}{3}$ and $h_t(3) = 0.2$. User 2 receives about four times the allocation of user 3 if ε is small enough. However, a better allocation for users 2 and 3 is $\frac{\beta(2)}{\beta(2)+\beta(3)} = \frac{3}{5}$ and $\frac{\beta(3)}{\beta(2)+\beta(3)} = \frac{2}{5}$, respectively. These allocations reflect the ratio $\beta(2)/\beta(3)$ between active users.

Given this, we propose a slight modification of Algorithm 1, shown in Algorithm 2. As before, the plan is always to benefit active users. However, this time, we boost active users slightly more if they fall behind their "proportional SLA" among active users. Intuitively, if there are n < N active users for a long period of time, the allocation of these active users should converge to their proportional share.

Algorithm 2 (Extended Multiplicative Weight Update Algorithm).

Input: Parameters $0 < \varepsilon \le \frac{1}{10}, 0 < \eta < \frac{1}{3}$.
1. Initialization: $\mathbf{h}_1$ any allocation over Δ_ε and $\lambda = \frac{\varepsilon^2}{8N}$.
2. **For** $t = 1, \dots, T$ **do**
3. Set allocation $\mathbf{h}_t$.
4. Read active and inactive users A_t and B_t. $A_t^1 = \left\{ i \in A_t : h_t(i) < (1 - \varepsilon)\frac{\beta(i)}{\sum_{j \in A_t} \beta(j)} \right\}$, $A_t^2 = A_t \backslash A_t^1$.
5. Set gain function $g_t(i) = \begin{cases} 1 + \lambda & i \in A_t^1 \\ 1 & i \in A_t^2 \\ 0 & i \in B_t \end{cases}$.
6. Update allocation: $\hat{h}_{t+1}(i) = h_t(i)e^{\eta g_t(i)}$, $\forall i$ and $\mathbf{h}_{t+1} = \pi_{\Delta_\varepsilon}(\widehat{\boldsymbol{h}}_{t+1})$

For technical reasons, the set A_t^1, the active users with allocation below their proportional share among active users at time t, has to be defined as $\left\{ i \in A_t : h_t(i) < (1 - \varepsilon)\frac{\beta(i)}{\sum_{j \in A_t} \beta(j)} \right\}$. The reason behind this choice is to ensure that if $A_t^1 \neq \emptyset$ and the resource is nearly fully utilized, that is, $\sum_{i\in A_t} h_t(i) \geq 1-\varepsilon$, then there is a different active user $j \neq i$ from which we can move allocation to i. This is fundamental in the proof of Theorem 6 below.

In terms of work maximization and SLA satisfaction, Algorithm 2 provides exactly the same guarantees as Algorithm 1.

Theorem 5. *Given loads $L_1, \ldots, L_T$, for any $\varepsilon > 0$ and $\eta > 0$ such that $\varepsilon \leq 1/10$, Algorithm 2 guarantees*

$$\text{work}_{\mathbf{h}_1^*,\ldots,\mathbf{h}_t^*} - \text{work}_{alg,t} \leq 8\frac{N}{\varepsilon^2\eta}\ln(N/\varepsilon),$$

for any time $1 \leq t \leq T$, where $\mathbf{h}_1^*, \ldots, \mathbf{h}_T^*$ is an optimal offline sequence of $(1-4\varepsilon)$-allocations, and $\text{work}_{alg,t} = \sum_i \sum_{\tau=1}^t w_\tau(i)$ is the overall work done by Algorithm 2 until time t.

The proof of Theorem 5 is exactly the same as the proof of Theorem 3. To see this, observe that the proof of Theorem 3 uses only the fact that the allocations of every active user get a multiplicative boost whenever the usage is below $1-\varepsilon$. The last statement is true since Lemma 1 also holds in this case.

For SLA satisfaction, we have the following stronger statement.

Theorem 6. *Let $0 < \varepsilon \leq 1/10$, $0 < \eta \leq 1/3$, $\lambda = \varepsilon^2/(8N)$ and $\tilde{p} = 32N^2\varepsilon^{-3}\eta^{-1}\ln(N/\varepsilon)$. Take any SLAs $\beta(1), \ldots, \beta(N)$ such that $\frac{\beta(i)}{\sum_k \beta(k)} \geq e^{\eta(1+\varepsilon)}\frac{\varepsilon}{(1-\varepsilon)N}$. Then, for any user i and time t, if we take $\mathbf{h}_1', \ldots, \mathbf{h}_T'$ to be the allocations such that $h_t'(i) = (1-2\varepsilon)\frac{\beta(i)}{\sum_k \beta(k)}$, the work done by Algorithm 1 for user i satisfies*

$$\sum_{\tau=1}^{t+\tilde{p}} w_\tau(i) \geq \sum_{\tau=1}^{t} w_\tau'(i),$$

where $\mathbf{w}_t'$ is the work done by the allocations $\mathbf{h}_1', \ldots, \mathbf{h}_T'$. Moreover, $\sum_{\tau=1}^t w_\tau(i) \geq \sum_{\tau=1}^t w_\tau'(i) - \frac{\beta(i)}{\sum_k \beta(k)}\tilde{p}$.

The proof of this result is similar to the proof of Theorem 4. A subtle difference is that the analog of Lemma 2 holds if we add the hypothesis $\sum_{i\in A_t} h_t(i) > 1-\varepsilon$. We skip the proof for brevity.

Lemma 3. *Assume that $\varepsilon \leq 1/10$, $\eta \leq 1/3$ and $\frac{\beta(i)}{\sum_k \beta(k)} \geq e^{\eta(1+\varepsilon)}\frac{\varepsilon}{(1-\varepsilon)N}$ for all users. In Algorithm 2, if $\sum_{k\in A_t} h_t(k) > 1-\varepsilon$, then for any $i \in A_t^1$ we have $h_{t+1}(i) \geq (1+c')h_t(i)$, where $c' = \varepsilon\eta\lambda/(2N)$.*

Lemmas 1 and 3 ensure that any active user gets a multiplicative boost of at least $(1+c')$. Therefore, any user that is active $\tilde{p}$ consecutive times will have an allocation of at least $(1-2\varepsilon)\frac{\beta(i)}{\sum_k \beta(k)}$. Then, by following the same inductive proof of Theorem 4, we obtain Theorem 6.

If the resource is not overcommitted, $\sum_k \beta(k) \leq 1$ this result implies that Algorithm 2 ensures for each user i an amount of work comparable with $\frac{\beta(i)}{\sum_k \beta(k)} \geq \beta(i)$; that is, we obtain the standard SLA satisfaction guarantee. In the overcommitment regime, $\sum_{i=1}^N \beta(i) > 1$, we do retain some performance guarantees. The update according to almost-normalized SLAs in Algorithm 2 still works, and Theorem 5's work maximization guarantee still applies, as its proof does not rely on the SLAs. On the other hand, Theorem 6 states that individually, each user does work comparable to their normalized SLA. If the level of overcommitment is not large, each user is still guaranteed service "almost" at their SLA; for example, if the resource is overcommitted by 10%, each user receives service comparable to $1.1^{-1} \approx 91\%$ of their SLA.

Another interesting byproduct of the work maximization guarantee is the following result.

Corollary 3. *Under the assumptions of Theorem 4, suppose there is a time $1 \leq \tau \leq T$ with $\sum_{t=1}^{\tau}\sum_i w_\tau'(i) = \sum_{t=1}^{\tau}\sum_i L_t(i)$; that is,. the optimal offline $(1-4\varepsilon)$ -allocation is able to finish all work up until τ. Then, the sum of queue lengths at time τ induced by Algorithm 2 is at most $8N\varepsilon^{-2}\eta^{-1}\ln(N/\varepsilon)$. In particular, at time τ each user's queue length is at most this value.*

In practical settings, it is commonplace to assume that an arrival rate is lower than the work processing rate. In stochastic settings, stationary states cannot be achieved without this assumption; see, for example, Tassiulas and Ephremides (1990). In our context, this can be reinterpreted as having times within the operating horizon where the optimal offline solution is able to finish all work arriving up until that time. At these particular times, the corollary guarantees that Algorithm 2's queue lengths are constant. In other words, the algorithm does not starve individual users to achieve work maximization and keeps their queues short, an appealing property in cloud systems.

4. Experiments

In this section, we empirically test Algorithm 2 against a family of offline and online algorithms. We aim to measure the performance on both synthetic data as well as real CPU traces from a production service in Microsoft's cloud. We quantify the performance on the following three criteria:

- **Work maximization.** We compare the overall work done by Algorithm 2 against various benchmark algorithms.

- **SLA guarantee.** We examine the extent to which our algorithm achieves the cumulative work of the static SLA policy for each user. We do so by measuring the cumulative work over plausible time windows.
- **Queue length.** We compare the 2-norm of the individual queues over time. We use this metric as a proxy for the system latency, which is not captured by our theoretical results.

We consider the following online algorithms, against which we benchmark our algorithm:

- **Static SLA Policy (Static)**. Each user gets their SLA as a constant, static allocation. We call this algorithm Static.
- **Proportional Online (PO)**. In each iteration, every active user will get their SLA normalized by the sum of SLAs of active users (just their SLA if there are no active users). This simple algorithm seems suitable for a practical implementation; however, its performance can be arbitrarily bad in terms of work maximization. The formal description appears in Algorithm 4 in Appendix E. We call this Algorithm PO.
- **Online Work Maximizing (OWM)**. This algorithm divides users into three categories: A, B, and I (active users with allocation, actives users without allocation, and inactive users). At each iteration, the resource is divided uniformly among users in A. If a user in A becomes inactive, they are moved to I. If a user in I becomes active, they are moved to B. When A becomes empty, we move all users from B to A. In Appendix C, we prove that this method is work maximizing. However, this greedy strategy is not guaranteed to satisfy SLA constraints for general input loads. We call this algorithm OWM.

We also consider the following offline algorithms, against which we benchmark our algorithm.

- **Optimal 1-allocations (PG)**. The optimal offline solution to the work maximization problem. This solution is computed using Algorithm 3, which we call Proportional Greedy (PG). This algorithm can be considered as the offline counterpart of Proportional Online.
- **Optimal $(1-\varepsilon)$-allocation (restPG).** Offline solution to the work maximization problem with resource restricted to $1-\varepsilon$, where ε is the parameter of Algorithm 2.

4.1. Synthetic Experiment

4.1.1. Description of the Experiment. In this experiment, we consider a synthetically generated input sequence, which we use to examine how online algorithms adapt to different conditions. Specifically, our system consists of three users with SLAs of $\beta(1)=0.2$, $\beta(2)=0.3$ and $\beta(3)=0.5$. We consider a time horizon of $T=3{,}000{,}000$. The load input sequence is divided into six periods: $P_i = [(i-1)T/6, iT/6)$ for $i=1,\ldots,6$. In each period, only two users demand new resources. During the first three periods, the random demand has a mean proportional to the users' SLA. In the following three periods, the random demand changes to a distribution with uniform mean among users demanding resources. Specifically:

- During P_1, only users 2 and 3 demand the following loads. At the beginning of P_1, that is, $t=1$, user 2 demands a large load of $L_1(2)\sim \frac{T}{6}\cdot \text{Gamma}(2000, \frac{1}{2000}\cdot \frac{\beta(2)}{\beta(2)+\beta(3)})$ and $L_1(3)=0$. During the rest of period P_1, $L_t(2)=0$ and $L_t(3)\sim \text{Gamma}\left(2000, \frac{1}{2000}\cdot\frac{\beta(3)}{\beta(2)+\beta(3)}\right)$. User 1 demands nothing during this entire period. Similar loads are set for period P_2 and P_3.
- Similarly, during P_4, users 2 and 3 demand $L_t(i)\sim \text{Gamma}\left(2000, \frac{1}{2000}\cdot\frac{1}{2}\right)$ for $i=2,3$. During $P_,$ users 1 and 2 demand $L_t(i)\sim \text{Gamma}\left(2000, \frac{1}{2000}\cdot\frac{1}{2}\right)$ for $i=1,2$. During P_6, users 1 and 3 demand $L_t(i)\sim \text{Gamma}(2000, \frac{1}{2000}\cdot\frac{1}{2})$ for $i=1,3$.

The expectation of a gamma(k,θ) random variable is given by $k\theta$, and the variance is given by $k\theta^2$ (see, e.g., Feller 1957). For example, in period P_1, user 2's expected load is $\frac{T}{6}\frac{\beta(2)}{\beta(2)+\beta(3)}$, with variance $\frac{1}{2000}\left(\frac{\beta(2)}{\beta(2)+\beta(3)}\right)^2$. Similarly, user 3's expected total load is $\frac{T}{6}\frac{\beta(3)}{\beta(2)+\beta(3)}$. Thus, the expected overall load is $T/6$, exactly the length of the period. A brief summary of gamma distribution's properties is given in Appendix F.

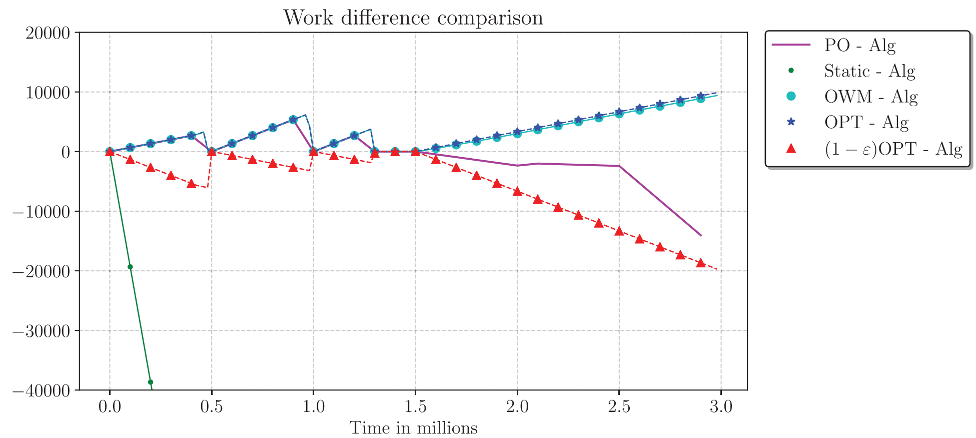
We instantiate Algorithm 2 with $\eta=\frac{1}{3}$, $\varepsilon=0.02$ and $T=3{,}000{,}000$.

4.1.2 Results

4.1.2.1. Work Maximization. In Figure 3, we present the cumulative work difference between PG and Algorithm 2 (solid blue line with star), restPG and Algorithm 2 (red dashed line with triangle), PO and Algorithm 2 (solid magenta line), static and Algorithm 2 (solid green line with small circle), and OWM and Algorithm 2 (solid cyan line with large circle). Intuitively, one positive unit of difference implies the corresponding algorithm is ahead of Algorithm 2 by one unit of time.

First, we consider the comparison with online algorithms static, PO, and OWM. Algorithm 2 outperforms Static significantly, by roughly 700,000 units of time. During the first half of the experiment, PO shows good performance, but in the second half of the experiment (when the load distribution changes), Algorithm 2 outperforms PO. This shows that Algorithm 2 can adapt to changing input sequences that PO cannot adapt to. Finally, OWM surpasses Algorithm 2 during the whole experiment,

Figure 3. (Color online) Difference Between Algorithms' Work



Notes. Alg is short for Algorithm 2. Observe that the differences "OPT - Alg" and "OWM - Alg" slightly overlap.

with a performance similar to PG; this is an expected result since OWM is work maximizing.

With respect to offline algorithms, PG outperforms Algorithm 2 by roughly 10,000 time units. On the other hand, Algorithm 2 surpasses restPG by approximately 20,000 units. This shows that Algorithm 2 indeed performs better than the offline optimum with a slightly reduced amount of resources, as guaranteed by the theoretical results.

4.1.2.2. SLA Satisfaction. Among online algorithms, static and PO satisfy SLA restrictions by design, but as seen earlier, they are not competitive in terms of work maximization. We focus on the comparison between OWM and Algorithm 2 as far as SLA satisfaction is concerned. Although OWM performs extremely well in work maximization, this comes at a significant price in SLA satisfaction. In Figure 4, we depict empirically this behavior by plotting the instantaneous work done by users 2 and 3 by OWM and Algorithm 2 during period P_1. We empirically observe that Algorithm 2 approximately satisfies user 3's SLA, but OWM does not allocate the user any resources. Such an extreme behavior arises because OWM is geared toward work maximization rather than SLA satisfaction.

4.1.2.3. Queue Lengths. In Figure 5, we present the 2-norm of queues induced by Algorithm 2 (solid blue), static (solid magenta with star), PO (solid cyan with large circle), and OWM (solid black with triangle). Once again, we can interpret one unit of norm as one unit of latency. Experimentally, we observe that static shows the worst performance with a final 2-norm of 524,000 units. Algorithm 2 ends with a 2-norm of 10,000 units, PO with 26,970 units, and OWM with 381 units. As remarked above, even though OWM induces very small queues, this comes at the cost of not satisfying SLA requirements.

4.1.2.4. Summary. Among all online algorithms, Algorithm 2 is able to best balance work maximization and SLA satisfaction. In particular, Algorithm 2 is only slightly worse in terms of work maximization compared with OWM; this is expected since Algorithm 2 always reserves a small fraction of the resource for each user, regardless of activity. Furthermore, our results show that the actual total work done by Algorithm 2 is much better than the theoretical guarantee of Theorem 5, because it substantially outperforms $(1-\varepsilon)$ OPT (restPG). Finally, we observe small queues throughout the entire horizon, which is key for maintaining reasonable latencies.

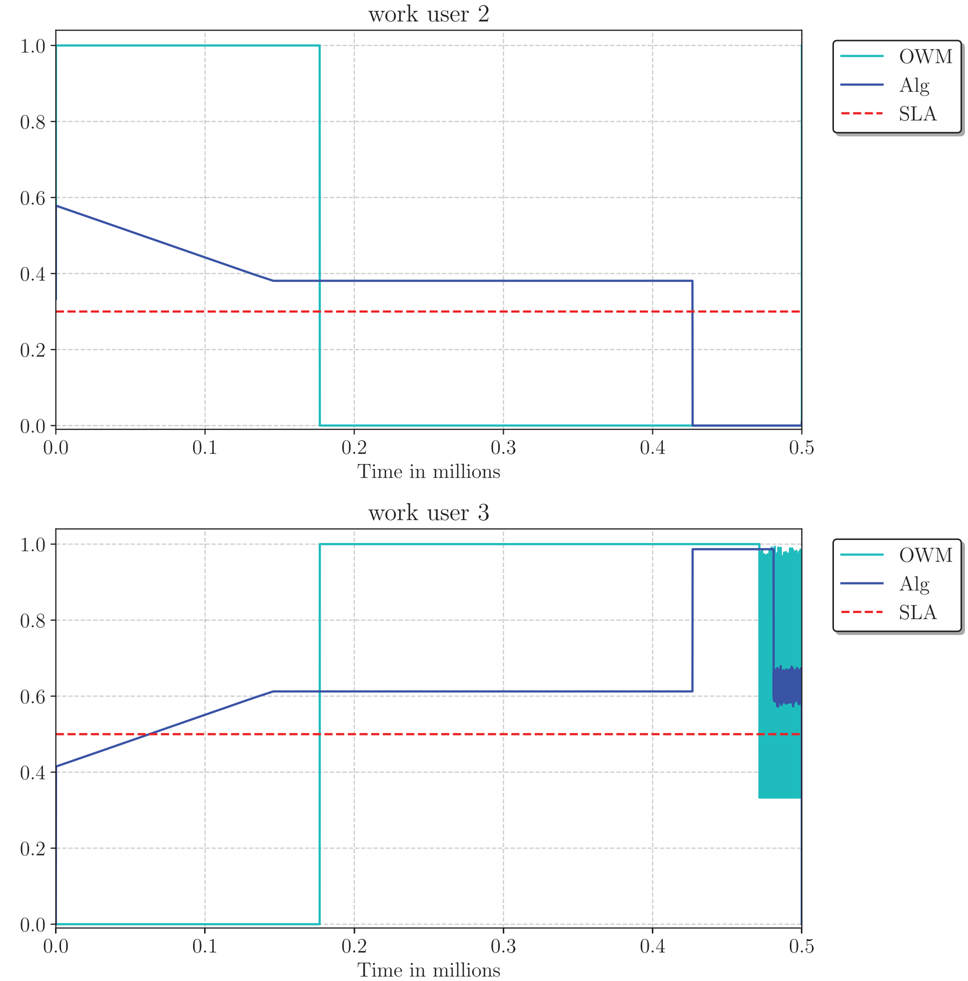
4.2. Experiment With Real Data

We next describe the results of our computational study using real-world data. For these experiments, we obtained CPU traces of a production service on Azure, Microsoft's public cloud. The data consists of demand traces of six different users over a time window of approximately 10 days.

To show Algorithm 2's robustness with respect to (short-term) real data, we also consider the following measurement:

- **Instantaneous SLA.** We focus on a modified SLA satisfaction criterion because of the relatively short horizon (about 14,000 minutes); we assess Algorithm 2's performance in the following way. For a user i and any time t, we compare the cumulative work done by user i in Algorithm 2 during a time window $[t, t+\tau)$ versus the cumulative work done by the same user i under a static SLA policy during the time window $[t, t+\tau)$. In order to have a meaningful comparison, at time t, we run the static SLA policy with the queues of Algorithm 2 at time t. The motivating question is, "*What happens if at time t and the next τ time steps we run the static policy instead of Algorithm 2?*" For this experiment, we used $\tau =$ 500 minutes.

Figure 4. (Color online) Instantaneous Work for Users 2 and 3 During period P_1



Note. User 3 does not receive any allocation in OWM until user 2 finishes all of his or her work.

Figure 5. (Color online) 2-Norm of Queues

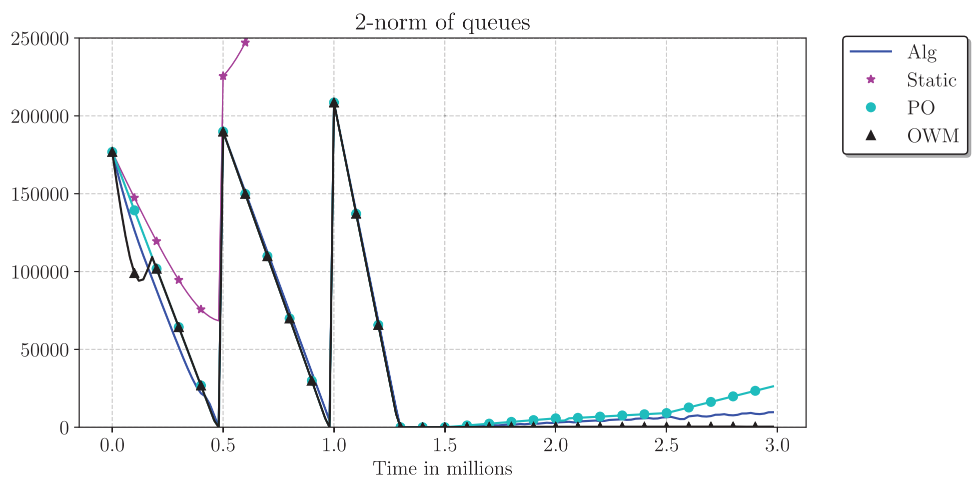
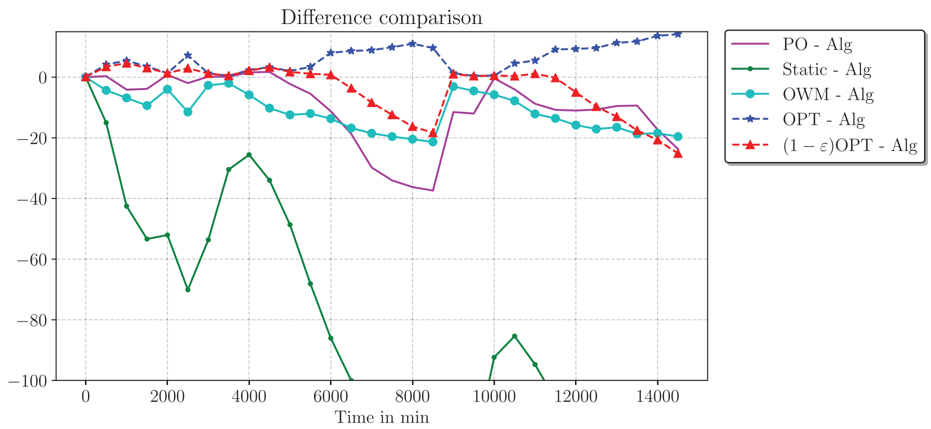


Figure 6. (Color online) Difference of Cumulative Works



The dataset consists of demand traces of six users of exactly 14,628 minutes (approximately 10 days). Each user is assigned their normalized average workload as SLA. (Because the data are proprietary, we cannot disclose actual SLAs.) For the purpose of the experiment, we run Algorithm 2 with parameters $\varepsilon = 0.01$ and $\eta = \frac{1}{3}$.

4.2.1. Results. ***4.2.1.1. Work Maximization.*** We depict in Figure 6 the following differences: cumulative work done until time t by optimal 1-allocations (PG) and Algorithm 2, $(1-\varepsilon)$-allocations (restPG) and Algorithm 2, PO and Algorithm 2, static and Algorithm 2, and OWM and Algorithm 2. In a similar fashion to the previous experiment, one positive unit can be interpreted as Algorithm 2 being one unit (minute) of work behind, and one negative unit means Algorithm 2 is ahead by a minute.

We observe that Algorithm 2 outperforms all online benchmarks. Against static, the final difference is 105 units, with a maximum difference of 167 units. Against PO, the final difference is 25 units, with a maximum difference of 37. Finally, against OWM, the final difference is 20, with a maximum difference of 21. Surprisingly, for this dataset, Algorithm 2 is able to surpass even OWM.

Regarding the offline algorithms, PG surpasses Algorithm 2 during the whole experiment as expected, with a final difference of 14 units. On the other hand, Algorithm 2 outperforms restPG by 26 units by the end of the experiment.

4.2.1.2. Instantaneous SLA. In Table 1, we report statistics on the differences between the cumulative work done in time windows $[t, t+\tau)$ by static and Algorithm 2 for each user. For each t and each user i, the exact formula is $r_i(t) = \sum_{r=t}^{t+\tau-1} w_r'(i) - \sum_{r=t}^{t+\tau-1} w_r(i)$, where $w_r(i)$ is the work done by user i under Algorithm 2 and $w_r'(i)$ is the work done by user i with static (with queues at t given by Algorithm 2). The table shows the minimum, maximum, average, and standard deviation of $\{r_i(t)\}_t$ when $\tau = 500$ minutes. A positive unit means Algorithm 2 is outperformed by static during $[t, t+\tau)$ under the same initial conditions by one unit of time. In general, we observe that all users show negative empirical average difference. This result empirically suggests that Algorithm 2 ensures approximate SLA satisfaction, even for small time windows. For instance, user 3 occasionally has a high difference (46.4 units), mostly due to times t where Algorithm 2 allocates the user a small amount of resource but a huge load is incoming during the window $[t, t+\tau)$. The experiment tells us that averaging out these "bad" times ensures good performance under the SLA criterion. Furthermore, we tested values of τ = 60, 500, and 1,000 minutes; larger windows improve the results, with lower maximum and average values.

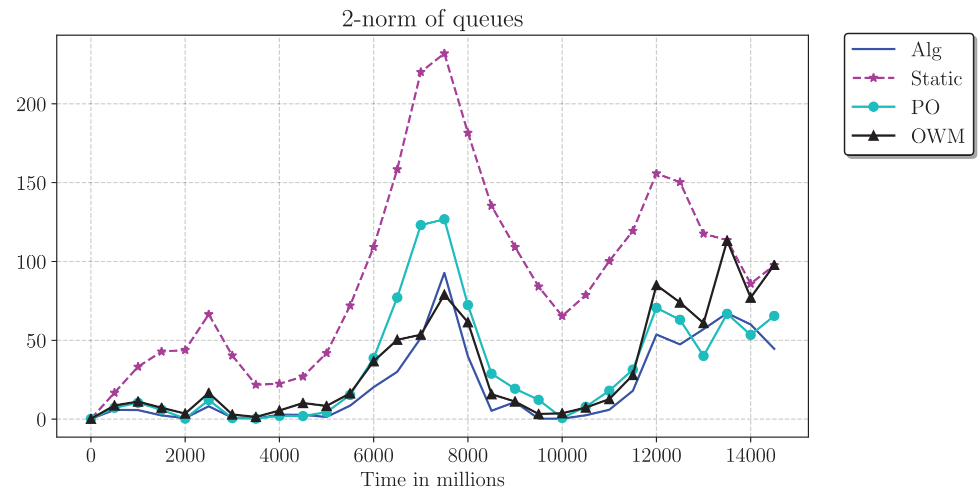
4.2.1.3. Queue Lengths. In Figure 7, we present the 2-norms of queues given by the online benchmark

Table 1. Statistics for the Difference Between the Cumulative Works of Our Algorithm and Static over Time Windows $[t, t+\tau)$

User	Minimum	Maximum	Mean	SD
User 1	−31.1	21.7	−4.6	13.8
User 2	−122.3	46.9	−31.2	49.2
User 3	−90.8	46.4	−7.2	29.5
User 4	−49.7	18.5	−0.8	12.9
User 5	−42.6	22.6	−5.85	14.0
User 6	−21.9	14.9	−0.6	6.4

Note. For each user i we present the minimum, maximum, average, and standard deviation over the sequence of differences $\{r_i(t)\}_t$.

Figure 7. (Color online) 2-Norm of Queues



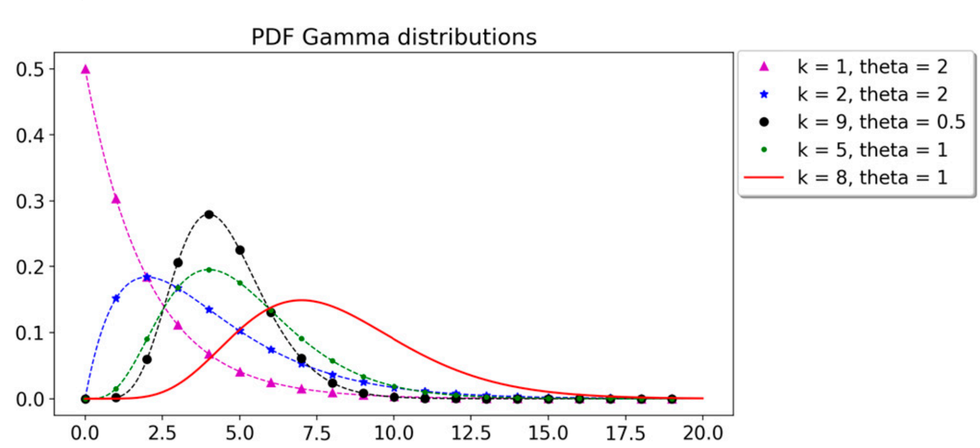
algorithms and Algorithm 2. As usual, we can interpret one unit as the respective algorithm's latency, that is, lateness with respect to the overall users' demand. Compared against the online algorithms, we empirically observe the superiority of Algorithm 2, because it has the smallest latency most of the time. Algorithm 2 ends with a 2-norm of roughly 44 units, an average length of 22 and a maximum length of 92. PO ends with a 2-norm of approximately 68, an average length of 32 and a maximum length of 126. OWM ends with a 2-norm of 68, an average of 32 and maximum of 113. Finally, static shows the worst behavior, with a final 2-norm of 103, an average of 91 and maximum of 231. For this dataset, Algorithm 2 shows a remarkable performance, considering particularly that Algorithm 2 always reserves ε/N resource for each user.

4.2.1.4. Summary. Algorithm 2 performs very well in terms of work maximization compared with all other online algorithms and exceeds the theoretical guarantees. Furthermore, SLA requirements are typically satisfied, even when measured over relatively short time windows. Finally, as in the previous experiment, the algorithm maintains small queues compared with other online algorithms.

5. Conclusion

We have proposed a new online model for dynamic resource allocation of a single divisible resource in a shared system. Our framework captures basic properties of cloud systems, including SLAs, limited system feedback, and unpredictable (even adversarial) input sequences. We designed an algorithm that is near-optimal in terms of both work maximization and SLA satisfaction (Theorems 1, 5, and 6). Furthermore, our second algorithm, Algorithm 2, can be applied in an overcommitment regime with similar guarantees, which could be of additional merit for some applications. We derived a simple expression for the offline

Figure 8. (Color online) PDF of Different Gamma Distributions



work maximization problem that allowed us to reinterpret the algorithm's dynamics as an approximate solution of the optimal (offline) work maximization LP. Numerical experiments show that our algorithm is indeed able to achieve a desirable trade-off between work maximization and SLA satisfaction. In particular, comparisons with offline algorithms (PG and restPG) indicate that our algorithm is empirically work maximizing. Furthermore, unlike other plausible online algorithms, our algorithm is able to quickly adapt to unexpected changes in demand and still approximately satisfy the underlying user SLAs. Our model and results may be extended in various directions of interest to the operations research and cloud computing communities.

A natural extension for single-resource systems is to model priority among users. Typically, users with higher priority should be given resources before their lower-priority counterparts. A challenge in this setting is how to define the metric corresponding to work maximization. One possibility is to have different weights for different users, corresponding to their priority, and then to maximize the weighted total work while satisfying SLAs. Directly extending our algorithms to this case means a user's multiplicative boost depends on their priority. However, our analysis in this paper does not apply because we use the fact that users' work is interchangeable, whereas the identity of who performs the work is critical in the prioritized case.

Another challenging extension is the management of multiple resources (e.g., CPU, I/O bandwidth, memory), where different users or jobs may require the resources in different proportions. This extension requires a fundamental redefinition of our model, where work done for a user is a function of the multiple resources allocated and may also depend on a particular job's characteristics. In many real-world scenarios, a job's resource demands are often complementary, for example, RAM and CPU usage. This observation may motivate a possible extension in which we still treat all users' loads as one-dimensional quantities, and the work performed by a user is a relatively simple function of their allocations, for example, a concave nondecreasing function.

Acknowledgment
The authors thank Vivek Narasayya for useful discussions.

Appendix A: Projecting on Δ_ε

Proof of Proposition 1. Let $\mathbf{y} \in \mathbf{R}_+^N$. The projection of $\mathbf{y}$ on Δ_ε corresponds to the solution of the convex problem

$$(Q) \qquad \begin{array}{lrcl} \min & \sum_i x(i)\ln\left(\frac{x(i)}{y(i)}\right) & & \\ & \sum_i x(i) & = & 1 \\ & x(i) & \geq & \varepsilon/N \end{array}$$

Its Lagrangian (see Boyd and Vandenberghe 2004) is

$$\mathcal{L}(\mathbf{x},\lambda,\mu) = \sum_i x(i)\ln\left(\frac{x(i)}{y(i)}\right) - \lambda\left(\sum_i x(i) - 1\right) - \sum_i \mu_i\left(x(i) - \frac{\varepsilon}{N}\right).$$

Using the FO conditions,

$$\forall i : x(i) = y(i)e^{\mu_i+\lambda-1} = y(i)e^{\mu_i}C.$$

and the SO conditions,

$$\mu_i \geq 0,\ \forall i, \quad \text{and} \quad x(i) > \frac{\varepsilon}{N} \Rightarrow \mu_i = 0.$$

Let $S = \{i : x(i) = \varepsilon/N\}$ and $T = [N]\backslash S$. Then, using $\sum_i x(i) = 1$, we obtain

$$e^{\lambda-1} = \frac{1-\frac{\varepsilon}{N}|S|}{\sum_{i\in T} y(i)}.$$

This proves part (b). Now, suppose we have $y(1) \leq \cdots \leq y(N)$. If $i,j \in T$, then $x(i) = y(i)e^{\lambda-1}$ and $x(j) = y(j)e^{\lambda-1}$, and then

$$x(i) \leq x(j) \Longleftrightarrow y(i) \leq y(j).$$

That is, in T, the variables preserve their ordering.

If $i \in S$ and $j \in T$, then $y(i)e^{\lambda-1+\mu_i} = x(i) = \varepsilon/N < x(j) = y(j)e^{\lambda-1}$, which implies $y(i) < y(j)$ using that $\mu_i \geq 0$. Now, let $k = \min\{i \in T\}$, which is a well-defined number using constraint $\sum_{i=1}^N x(i) = 1$. We claim that for any $j \geq k$, $j \in T$, that is, T corresponds to the interval $[k, N]$. By contradiction, suppose that $j > k$ does not belong to T; then, $y(j) < y(k)$ by previous calculus. However, $y(j) \geq y(k)$ by the ordering of $\mathbf{y}$, a contradiction. With this, the algorithm to project is clear, we sort $\mathbf{y}$, and then we test increasingly the possible set $S = \{1,\ldots,k-1\}$ for $k = 1,\ldots,N$ and select the best candidate. This proves (a).

We say that S is feasible if there is a feasible solution $\mathbf{x}$ such that $S = \{i : x(i) = \varepsilon/N\}$. In the following paragraphs, we prove that the first feasible solution found in this process is the right one.

Observe that once $S = \{1,\ldots,k\}$ is feasible, then $S' = \{1,\ldots,j\}$ remains feasible for all $j \geq k$. Indeed, if $S = \{1,\ldots,k\}$ is feasible, then

$$1 = \frac{\varepsilon}{N}k + \sum_{i\in T} x(i).$$

Now, increasing S to $S' = \{1,\ldots,k+1\}$ means that we pick $x(k+1) > \varepsilon/N$, and we decrease it to ε/N. Therefore, $x(k+2),\ldots,x(N)$ must increase. Therefore, S' remains feasible. The proof for general case $j \geq k$ follows by induction.

Now, we claim that if $S = \{1,\ldots,k\}$ is feasible, then $S' = \{1,\ldots,k+1\}$ cannot have better optimal value. Indeed, the difference between the objective S' and S is

$$\sum_{i=1}^{k+1}\frac{\varepsilon}{N}\ln\frac{\varepsilon}{Ny(i)} + \left(1-\frac{\varepsilon}{N}(k+1)\right)\ln\frac{1-\frac{\varepsilon}{N}(k+1)}{\sum_{i\geq k+2} y(i)} - \sum_{i=1}^{k}\frac{\varepsilon}{N}\ln\frac{\varepsilon}{Ny(i)}$$
$$-\left(1-\frac{\varepsilon}{N}k\right)\ln\frac{1-\frac{\varepsilon}{N}k}{\sum_{i\geq k+1} y(i)}$$
$$= \frac{\varepsilon}{N}\ln\frac{\varepsilon}{Ny(k+1)} + \left(1-\frac{\varepsilon}{N}(k+1)\right)\ln\frac{1-\frac{\varepsilon}{N}(k+1)}{\sum_{i\geq k+2} y(i)}$$
$$-\left(1-\frac{\varepsilon}{N}k\right)\ln\frac{1-\frac{\varepsilon}{N}k}{\sum_{i\geq k+1} y(i)}$$

The function $f(x) = x\ln x$ is convex for $x > 0$. Now, pick $x = \frac{\varepsilon}{Ny(k+1)}$, $y = \frac{1-\frac{\varepsilon}{N}(k+1)}{\sum_{i\geq k+2} y(i)}$, and $\lambda = \frac{y(k+1)}{\sum_{i\geq k+1} y(i)}$. Then

$$\lambda x + (1-\lambda)y = \frac{y(k+1)}{\sum\limits_{i\geq k+1} y(i)}\left(\frac{\varepsilon}{Ny(k+1)}\right) + \frac{\sum\limits_{i\geq k+2} y(i)}{\sum\limits_{i\geq k+1} y(i)}\left(\frac{1-\frac{\varepsilon}{N}(k+1)}{\sum\limits_{i\geq k+2} y(i)}\right) = \frac{1-\frac{\varepsilon}{N}k}{\sum\limits_{i\geq k+1} y(i)}.$$

Then, using the convexity of f, we obtain the result. This implies that the first feasible prefix S that we find is the optimal one. Therefore, by ordering $\mathbf{y}$ in $\mathcal{O}(N\log N)$ time and then running a binary search, we can find S in $\mathcal{O}(N\log N)$ time. This finishes the proof of (c). $\square$

Appendix B: Omitted Proofs

B.1 Proofs of Section 3.1

Here, we present dual stated in the offline formulation of the maximum work problem. We have the LP

$$(P_\varepsilon)\quad \begin{array}{llll} \max \sum_{i=1}^N \sum_{t=1}^T w_t(i) & & & \\ \sum_{s=1}^t w_s(i) \leq \sum_{s=1}^t L_s(i) & \forall t, i & (1) \\ \sum_{i=1}^N w_t(i) \leq 1-\varepsilon & \forall t & (2) \\ \mathbf{w}_t \geq 0 & \forall t & \end{array}$$

Using the variables $\alpha_t(i)$ for constraint (1) and β_t for constraint (2), we obtain the dual

$$(D_\varepsilon)\quad \begin{array}{lll} \min \sum_{i=1}^N \sum_{t=1}^T \alpha_t(i) \sum_{s=1}^t L_s(i) + (1-\varepsilon)\sum_{t=1}^T \beta_t & & \\ \sum_{s=t}^T \alpha_s(i) + \beta_t \geq 1 & \forall t, i & (1') \\ \alpha, \beta \geq 0 & & \end{array}$$

Using the change of variable $\gamma_t(i) = \sum_{s=t}^T \alpha_s(i)$, we obtain the stated dual

$$(D_\varepsilon)\quad \begin{array}{lll} \min \sum_{i=1}^N \sum_{t=1}^T L_t(i)\gamma_t(i) + (1-\varepsilon)\sum_{t=1}^T \beta_t & & \\ \gamma_t(i) + \beta_t \geq 1 & \forall t, i & (1') \\ \gamma_t \geq \gamma_{t+1} & \forall t & (2') \\ \beta, \gamma \geq 0 & & \end{array}$$

Proof of Proposition 2. We prove each inequality separately. Let $0 \leq t^\star \leq T$ be such that $\sum_{s=1}^{t^\star}\sum_i L_s(i) + (1-\varepsilon)(T-t^\star) = \min_{0\leq t\leq T}\sum_{s=1}^t \sum_i L_s(i) + (1-\varepsilon)(T-t)$. Consider the dual solution (β, γ) such that $\gamma_t = 1$, $\beta_t = 0$ for $t = 1, \ldots, t^\star$, and $\gamma_t = 0, \beta_t = 1$ for $t = t^\star + 1, \ldots, T$. Then, by weak duality,

$$v_{P_\varepsilon} \leq v_{\text{dual}}(\beta,\gamma) = \sum_{s=1}^{t^\star}\sum_i L_s(i) + (1-\varepsilon)(T-t^\star) = \min_{0\leq t\leq T}\sum_{s=1}^t \sum_i L_s(i) + (1-\varepsilon)(T-t).$$

Now, consider the greedy algorithm that in each iteration gives enough allocation to the users in order to complete their work, starting with user 1, then user 2, and so on. We restrict the algorithms' allocations to $(1-\varepsilon)$-allocations. We denote by $\text{work}_{\text{greedy}}$ the work done by this algorithm. As usual, we denote by $\mathbf{w}_t$ the vector of work done at time t. Let $t^\star$ be the maximum nonnegative t such that $\sum_i w_t(i) < 1-\varepsilon$. Observe that $\sum_{s=1}^{t^\star}\sum_i w_s(i) = \sum_{s=1}^{t^\star}\sum_i L_s(i)$. Then

$$\min_{0\leq t\leq T}\sum_{s=1}^t\sum_i L_s(i) + (1-\varepsilon)(T-t) \leq \sum_{s=1}^{t^\star}\sum_i L_s(i) + (1-\varepsilon)(T-t^\star) = \text{work}_{\text{greedy}} \leq v_{P_\varepsilon},$$

since v_{P_ε} is the optimal solution. $\square$

Remark 1. This max-min result shows that the greedy algorithm is optimal for solving (P_ε) and also shows how to compute the dual variables. Finally, solving (P_ε) can be done efficiently in $\mathcal{O}(NT)$ by running the greedy algorithm.

B.2 Proofs of Section 3.2

In what follows, we denote by S_t the users with allocation $\frac{\varepsilon}{N}$ at time t.

Proof of Lemma 1.

1. First, for $i \in A_t$ we have

$$h_{t+1}(i) = \frac{h_t(i)e^{\eta g_t(i)}e^{\mu_i}(1-\frac{\varepsilon}{N}|S_{t+1}|)}{\sum_{j\in\bar{S}_{t+1}}\hat{h}_{t+1}(j)} \geq h_t(i)\frac{(1-\frac{\varepsilon}{N}|S_{t+1}|)}{e^{-\eta}\sum_{j\in\bar{S}_{t+1}}\hat{h}_{t+1}(j)}.$$

We divide the analysis into two cases: $B_t \cap \bar{S}_{t+1} \neq \emptyset$ and $B_t \subseteq S_{t+1}$.
For $B_t \cap \bar{S}_{t+1} \neq \emptyset$ we have

$$\begin{aligned} e^{-\eta}\sum_{j\in\bar{S}_{t+1}}\hat{h}_{t+1}(j) &\leq e^{\lambda\eta}\sum_{j\in\bar{S}_{t+1}\cap A_t} h_t(j) + e^{-\eta}\sum_{j\in\bar{S}_{t+1}\cap B_t} h_t(j) \\ &= e^{\lambda\eta}\sum_{j\in\bar{S}_{t+1}} h_t(j) - (e^{\lambda\eta} - e^{-\eta})\sum_{j\in B_t\cap\bar{S}_{t+1}} h_t(j) \\ &\leq e^{\lambda\eta}\left(1-\frac{\varepsilon}{N}|S_{t+1}|\right) - \frac{\varepsilon}{N}(e^{\lambda\eta}-e^{-\eta}) \,(\text{since } h_t(i) \geq \frac{\varepsilon}{N}) \\ &\leq (1+2\lambda\eta)\left(1-\frac{\varepsilon}{N}|S_{t+1}|\right) - \frac{\varepsilon}{N}\left(\lambda\eta + \eta - \frac{\eta^2}{2}\right) \\ &\leq 1 - \frac{\varepsilon}{N}|S_{t+1}| + 2\lambda\eta - \frac{\varepsilon}{N}\eta\left(1-\frac{\eta}{2}\right) \\ &\qquad (\text{using } 1+\lambda\eta \leq e^{\lambda\eta} \leq 1 + 2\lambda\eta, e^{-\eta} \leq 1 - \eta + \frac{\eta^2}{2}) \\ &\leq 1 - \frac{\varepsilon}{N}|S_{t+1}| - \frac{\varepsilon\eta}{4N}. \eta \leq 1 \text{ and } 2\lambda \leq \frac{\varepsilon}{4N} \end{aligned}$$

Therefore,

$$\frac{(1-\frac{\varepsilon}{N}|S_{t+1}|)}{e^{-\eta}\sum_{j\in\bar{S}_{t+1}}\hat{h}_{t+1}(j)} \geq \frac{1-\frac{\varepsilon}{N}|S_{t+1}|}{1-\frac{\varepsilon}{N}|S_{t+1}| - \frac{\varepsilon\eta}{4N}} \geq \frac{1}{1-\frac{\varepsilon\eta}{4N}} \geq 1 + \frac{\varepsilon\eta}{4N},$$

using $\frac{1}{1-x} \geq 1 + x$ when $x \in (0,1)$. Hence, $h_{t+1}(i) \geq h_t(i)(1+\varepsilon\eta/(4N))$.
Now, if $B_t \subseteq S_{t+1}$, then $\bar{S}_{t+1} \subseteq A_t$. We have

$$\begin{aligned} \frac{e^{-\eta}\sum_{j\in\bar{S}_{t+1}}\hat{h}_{t+1}(j)}{1-\frac{\varepsilon}{N}|S_{t+1}|} &\leq \frac{(1-\varepsilon)e^{\lambda\eta}}{1-\frac{\varepsilon}{N}|S_{t+1}|} \\ &\leq \frac{(1-\varepsilon)e^{\lambda\eta}}{1-\varepsilon+\frac{\varepsilon}{N}} \\ &\leq \frac{(1-\varepsilon)(1+2\lambda\eta)}{1-\varepsilon+\frac{\varepsilon}{N}} \qquad (e^{\lambda\eta} \leq 1 + 2\lambda\eta \text{ since } 2\lambda\eta < 1) \\ &\leq \frac{1-\varepsilon+3\lambda\eta}{1-\varepsilon+\frac{\varepsilon}{N}} = 1 - \frac{\frac{\varepsilon}{N}+3\lambda\eta}{1-\varepsilon+\frac{\varepsilon}{N}}. \end{aligned}$$

Because $\frac{\frac{\varepsilon}{N}+3\lambda\eta}{1-\varepsilon+\frac{\varepsilon}{N}} \geq \varepsilon/N$, we obtain

$$h_{t+1}(i) \geq h_t(i)\frac{1}{1-\frac{\varepsilon}{N}} \geq h_t(i)\left(1+\frac{\varepsilon}{N}\right) \geq h_t(i)\left(1+\frac{\varepsilon\eta}{4N}\right).$$

2. The monotonicity of $h_t(i)$ with $i \in A_t^1$ is easy to see. Let us prove the second statement:

$$\begin{aligned}\frac{e^{-\eta}\sum_{j\in\bar{S}_{t+1}}\hat{h}_{t+1}(j)}{1-\frac{\varepsilon}{N}|S_{t+1}|} &\leq e^{\lambda\eta}\frac{\sum_{j\in\bar{S}_{t+1}}h_t(j)}{1-\frac{\varepsilon}{N}|S_{t+1}|}\\ &\leq \frac{e^{\lambda\eta}\left(1-\frac{\varepsilon}{N}|S_{t+1}|\right)}{1-\frac{\varepsilon}{N}|S_{t+1}|}\\ &\leq e^{\lambda\eta}\\ &\leq 1+2\lambda\eta = 1+\varepsilon c,\end{aligned}$$

since $\lambda = \frac{\varepsilon^2}{8N}$. Then, for $i \in A_t^2$,

$$h_{t+1}(i) \geq h_t(i)\frac{e^{\eta}(1-\frac{\varepsilon}{N}|S_{t+1}|)}{\sum_{j\in\bar{S}_{t+1}}\hat{h}_{t+1}(j)} \geq h_t(i)\frac{1}{1+\varepsilon c} \geq h_t(i)(1-\varepsilon c). \quad \square$$

Proof of Claim 2. Now, let $[s^*+1,\ldots,T]$, and let us divide this interval into blocks of length $\tilde{s}$ with a possible last piece of length at most $\tilde{s}$. Let L be one of these blocks, and let i be the user given by claim 1, that is, $M_{i,r}=1$ for all $r \in L$. Consider $L' = \{t \in L : \sum_{j\in A_t} h_t(j) < 1-\varepsilon\}$. By using part 1 of Lemma 1, user i increases his or her allocation multiplicatively in L' by a factor of $(1+c)$. Observe that for $t \notin L'$, user's i allocation can increase or decrease, depending on $h_t(i)$. However, by lemma by part 2 of 1, we know that $h_t(i)$ will not decrease by a huge amount. Let $k' = |L'|$, and then i increases his or her allocation for k' times and decreases it for at most $\tilde{s}-k'$ times. Therefore, k' maximum value is such that

$$\frac{\varepsilon}{N}(1+c)^{k'}(1-\varepsilon c)^{\widetilde{s-k'}} = 1,$$

and therefore,

$$\begin{aligned}k' &\leq \frac{\ln(N/\varepsilon)+\tilde{s}\ln(1-\varepsilon c)^{-1}}{\ln((1+c)/(1-\varepsilon c))}\\ &\leq \frac{1+c}{c(1+\varepsilon)}\ln(N/\varepsilon)+\frac{\varepsilon(1+c)}{(1+\varepsilon)(1-\varepsilon c)}\tilde{s}\\ &\quad\left(\ln\frac{1+c}{1-\varepsilon c} \geq \frac{c(1+\varepsilon)}{1+c}, \ln\frac{1}{1-\varepsilon c} \leq \frac{\varepsilon c}{1-\varepsilon c}\right)\\ &\leq \frac{\varepsilon(1+c)}{(1+\varepsilon)}\tilde{s}+\frac{\varepsilon(1+c)}{(1+\varepsilon)(1-\varepsilon c)}\tilde{s} \qquad \left(\tilde{s} = \frac{\ln(N/\varepsilon)}{\varepsilon c}\right)\\ &= \varepsilon\frac{1+c}{1+\varepsilon}\left(1+\frac{1}{1-\varepsilon c}\right)\tilde{s}\\ &\leq 3\varepsilon\tilde{s}. \qquad \left(\text{for } N \geq 2 \text{ and } \varepsilon \leq \frac{1}{10}\right)\end{aligned}$$

Hence, L' is at most a fraction of $\tilde{s}$, and with this,

$$\sum_{t\in L}\sum_i w_t(i) \geq (1-\varepsilon)(\tilde{s}-k') \geq (1-\varepsilon)(1-3\varepsilon)\tilde{s} \geq (1-4\varepsilon)|L|.$$

Summing over all blocks we conclude the desired result. $\square$

B.3 Proof of Section 3.3

Proof of Lemma 2. If $A_t^1 = \emptyset$, the result is vacuously true. Suppose that $A_t^1 \neq \emptyset$. First, we prove that under the assumption of Lemma 2, we have $\bar{A}_t^1 \cap \bar{S}_{t+1} \neq \emptyset$. For $j \in S_{t+1}$, we have

$$\begin{aligned}\frac{\varepsilon}{N} &= h_{t+1}(j)\\ &= \frac{\hat{h}_{t+1}(j)e^{\mu_i}(1-\frac{\varepsilon}{N}|S_{t+1}|)}{\sum_{k\in\bar{S}_{t+1}}\hat{h}_{t+1}(k)} \quad (\text{since } \mu_i \geq 0)\\ &\geq \frac{h_t(j)(1-\frac{\varepsilon}{N}|S_{t+1}|)}{\sum_{k\in\bar{S}_{t+1}}h_t(k)e^{\eta(1+\lambda)}},\end{aligned}$$

This implies that $h_t(j) \leq \frac{\varepsilon}{N}e^{\eta(1+\lambda)} < \beta(j)$. Because $\sum_j h_t(j) = 1$, $i \in A_t^1$, and $\sum_j \beta(j) = 1$, we must have that there is a user $j \neq i$ with allocation $h_t(j) \geq \beta(j)$. Clearly, $j \notin S_{t+1}$ and $j \notin A_t^1$. Therefore, $\bar{A}_t^1 \cap \bar{S}_{t+1} \neq \emptyset$.

Following the proof of Lemma 1, for $i \in A_t^1$, we have

$$\begin{aligned}e^{-\eta(1+\lambda)}\sum_{j\in\bar{S}_{t+1}}\hat{h}_{t+1}(j) &\leq \sum_{j\in\bar{S}_{t+1}\cap A_t^1}h_t(j)+e^{-\eta\lambda}\sum_{j\in\bar{S}_{t+1}\cap\bar{A}_t1}h_t(j)\\ &= \sum_{j\in\bar{S}_{t+1}}h_t(j)-(1-e^{-\lambda\eta})\sum_{j\in\bar{S}_{t+1}\cap\bar{A}_t1}h_t(j)\\ &\leq 1-\frac{\varepsilon}{N}|S_{t+1}|-\frac{\varepsilon}{N}(1-e^{-\lambda\eta})\\ &\qquad(\text{since } \bar{S}_{t+1}\cap\bar{A}_t1 \neq \emptyset)\\ &\leq 1-\frac{\varepsilon}{N}|S_{t+1}|-\frac{\varepsilon\lambda\eta}{2N}\\ &\qquad\left(1-e^{-x} \geq \frac{x}{2} \text{ for } x \in [0,1]\right).\end{aligned}$$

Therefore,

$$h_{t+1}(i) \geq h_t(i)\frac{1-\frac{\varepsilon}{N}|S_{t+1}|}{1-\frac{\varepsilon}{N}|S_{t+1}|-\frac{\varepsilon\lambda\eta}{2N}} \geq h_t(i)\left(1+\frac{\varepsilon\lambda\eta}{2N}\right). \quad \square$$

Appendix C: Greedy Online Algorithm

In this section, we prove that the following greedy allocation strategy is almost optimal in work maximization. The algorithm divides the users into three categories: A, nonempty queue users with nonzero allocation; B, nonempty queue users with zero allocation; and I, empty queue users with zero allocation. At time t, a user $i \in A$ is left in A if he or she still has nonempty queue; otherwise we will move the user to I. A user $i \in I$ will be moved to B if that user's queue becomes nonempty; otherwise the user will remain in I. Finally, if all users from A are moved to I, then we will move all B to A; otherwise we will leave B untouched. In any case, we will distribute uniformly among the users that remain in A.

Users move from A to I, I to B, and B to A. Let $\mathbf{w}_t$ be the work done by the algorithm, and let $\mathbf{w}'_t$ be the optimal offline work.

Theorem 7. *For any loads $L_1,\ldots,L_T \in \mathbf{R}^N_{\geq 0}$ and any $\varepsilon > 0$, this greedy Algorithm guarantees*

$$\sum_{t=1}^{T}\sum_i w_t(i)+2\frac{N^2}{\varepsilon} \geq \sum_{t=1}^{T}\sum_i w'_t(i)$$

where $\mathbf{w}'_t$ is the work done by the optimal offline sequence of $(1-2\varepsilon/N)$-allocations.

Proof. Let t^* be the maximum $t \geq 0$ such that $\sum_{s=1}^{t+N^2}\sum_i w_s(i) \geq \sum_{s=1}^{t}\sum_i L_s(i)$. By claim 2, we know that each interval $[r, r+N^2/\varepsilon)$, with $r > t^\star$, has a user with a nonempty queue. As in claim 2, we divide the interval $[t^\star + 1, T]$ into blocks of length N^2/ε with a last block of length at most N^2/ε. Pick any of these blocks, say L, and let $L' = \{t \in L : \sum_i w_t(i) < 1\}$. It is easy to see that $|L'| \le 2N$, and therefore, summing over all block, we have $\sum_{t \ge t^\star+1} \sum_i w_t(i) + N^2/\varepsilon \ge (T - t^\star)(1 - 2\varepsilon/N)$. The conclusion follows applying weak duality to $(P_{2\varepsilon/N})$. □

Remark 2. Against the best 1-allocations, we can optimize ε and obtain $\varepsilon = \sqrt{N^3/T}$. This greedy strategy will be $\mathcal{O}(\sqrt{NT})$ far from the optimal dynamic work. Observe that this matches the lower bound in Theorem 2.

Appendix D: Lower Bound

Theorem 8. *For any online deterministic algorithm $\mathcal{A}$* setting at each time 1-allocations, with an underlying queuing system, and with the same limited feedback as Algorithm 1, there exists a sequence of online loads $L_1, \ldots, L_T$ such that $\mathrm{work}_{\mathbf{h}_1^*, \ldots, \mathbf{h}_T^*} - \mathrm{work}_{\mathcal{A}} = \Omega(\sqrt{T})$, where $\mathbf{h}_1^*, \ldots, \mathbf{h}_T^*$ are the optimal offline dynamic 1-allocations.

Proof. We consider the case with $N = 2$ users; the general case reduces to $N = 2$ by loading jobs only to two users. Let $\mathcal{A}$ be an online algorithm for allocating a divisible resource for 2 users and with underlying queuing system and limited feedback. Without loss of generality, we can assume that the allocations sets by $\mathcal{A}$ always add up to 1 at every time step.

We will construct a sequence of loads $L_t = (L_t(1), L_t(2))$ that at every time will add up to 1. This will ensure that the overall work done by the optimal offline dynamic policy will be T. On the other hand, we will show that this sequence of loads will lead to large queue length for at least one of the users. The main ingredient is to use the fact that the algorithm receives limited feedback about the state of the system, that is, which users have empty queue. In particular, this implies that if there are two distinct sets of load vectors L_t and L'_t for some interval $t \in [r, s]$ such that the queues remain nonempty on both these sequences, then the resource allocation to the users in the two load sequences must be identical.

We will divide the time window $[1, T]$ into phases. Each phase will begin with a configuration of queues, say, $Q = (Q(1), Q(2))$, where one of the queues is empty and the other one nonempty. We set $q = Q(1) + Q(2)$, and we denote by q_i the q at phase i. We define $q_0 = 0$. We will prove that at the end of each phase $i \ge 1$, $q_{i+1} \ge q_i + 1/4$, with all q_{i+1} cumulated in one queue and the other queue empty.

Initially, the algorithm has a fixed deterministic allocation $\mathbf{h}_1 = (h_1(1), h_1(2))$. If $h_1(1) \le h_1(2)$, then we load $L_1 = (0, 1)$. Otherwise, we load $L_1 = (1, 0)$. In any case, we have $q_1 \ge 1/2$ and all q_1 in one queue.

Now, we will describe how the general phases work. For the sake of simplicity, we will describe the phase starting at time $t = 1$. We have queue configuration $Q = (Q(1), Q(2))$ with $q > 0$. By the initial phase, we can assume that $q \ge 1/2$. Moreover, we can assume that only one of the queues is nonempty; this point will be clear after we describe how the phase works, and it is clearly true for phase 1. Phase i with $q = q_i$ will last at most $2q + 2$ time steps.

Suppose that $Q(1) = 0$ and $Q(2) > 0$. If $h_1(1) = 1$, then we load $L_1 = (0, 1)$, and the phase ends with q increased by 1 and user 1's queue empty. Therefore, we can assume that $h_1(1) < 1$. Our first load will be $L_1 = (h_1(1) + \varepsilon, h_2(2) - \varepsilon)$, with $0 < \varepsilon < \frac{1}{4}$ small enough and such that both queues are nonempty. The following loads will be $L_t = \mathbf{h}_t$, the allocation of $\mathcal{A}$ at time t. Observe that the first load will ensure that both users see nonempty queues until the end of the phase. Moreover, user 1 always has exactly ε remaining in his or her queue.

- If there is a time $\tau^\star \in [1, 2q+1]$ such that $h_{\tau^\star}(1) \ge 1/2$, then we change the load at time $\tau^\star$ for $L_{\tau^\star}{}' = (0, 1)$. This will increment q by at least $1/2 - \varepsilon \ge 1/4$, and the phase ends. Observe that $Q_{\tau^\star+1}(1) = 0$.
- We can assume now that for the loads $L_t = \mathbf{h}_t$ we always have $h_t(1) < 1/2$ for all $t \in [2, 2q+1]$. We change the loads to $L_t{}' = (1, 0)$ until time $\tau^\star$ in which user 2 empties his or her queue. Recall that the feedback of the algorithm is only the set of empty queues at every time step. Thus the behavior of $\mathcal{A}$ under L_t and $L_t{}'$ will be the same until time $\tau^\star$. Now, we change load $L_{\tau^\star}$ by $L_{\tau^\star}{}' = (1 - h_{\tau^\star}(2) + Q_{\tau^\star}(2) - \varepsilon', h_{\tau^\star}(2) - Q_{\tau^\star}(2) + \varepsilon')$ with $0 < \varepsilon' < 1/4$ small enough. This will ensure that queue 2 will be exactly ε'. Now, in an extra step, we load $L_{\tau^\star+1} = (1, 0)$. Again, we have q increased by at least $1/4$, and this ends the phase. Observe that $Q_{\tau^\star+2}(2) = 0$.

The analysis is similar for $Q(1) > 0$ and $Q(2) = 0$. Observe that at the end of each phase, only one queue is nonempty, and the other one is empty. In any case, we have the desired increment. With this, we can set the following recurrence, $q_0 = 0$, $q_{i-1} + 1/4 \le q_i \quad \forall i \ge 1$. We deduce that $q_i \ge i/4$. Now, let m be the number of phases. By construction, each phase lasts at most $2q_i + 2$. Then $T \le \sum_{i=1}^m (2q_i + 2) \le 40 q_m^2$, where we have used $4q_m \ge 4q_i \ge i \ge 1$. From here, we deduce that $q_m \ge \sqrt{T/40}$.

Now, the work done by the algorithm and the unfulfilled work in the queues must add up the overall load. Then $q_m + \mathrm{work}_{\mathcal{A}} = T = \mathrm{work}_{\mathbf{h}_1^*, \ldots, \mathbf{h}_T^*}$, from which we obtain the result. □

Appendix E: Additional Algorithms

Algorithm 3 (Proportional Greedy)

Input: Sequence of loads $(L_t)_{t=1}^T$ and SLAs $\beta(1), \ldots, \beta(N)$.

For $t = 1, \ldots, T$ **do**
    $w_t(i) \leftarrow 0, \forall i \in [N]$.
    $\mathrm{Rem} \leftarrow 1$.
    $\mathrm{Rem}(i) \leftarrow Q_{t-1}(i) + L_t(i), \forall i \in [N]$.
    Repeat
        $A \leftarrow \{i : \mathrm{Rem}(i) > 0\}$.
        $\Sigma \leftarrow \sum_{i \in A_t} \beta(i)$.
        $i^* \leftarrow \mathrm{argmin}_{k \in A} \mathrm{Rem}(k)$.
        If $\mathrm{Rem}(i) < \frac{\beta(i)}{\Sigma}\mathrm{Rem}$**, then**
            $w_t(i^*) \leftarrow w_t(i^*) + \mathrm{Rem}(i^*)$.
            $\mathrm{Rem} \leftarrow \mathrm{Rem} - \mathrm{Rem}(i^*)$.
            $\mathrm{Rem}(i^*) \leftarrow 0$.
        Else
            for $k \in A$**, do**
                $w_t(k) \leftarrow w_t(k) + \frac{\beta(k)}{\Sigma}\mathrm{Rem}$.
                $\mathrm{Rem}(k) \leftarrow \mathrm{Rem}(k) - \frac{\beta(k)}{\Sigma}\mathrm{Rem}$.
            $\mathrm{Rem} \leftarrow 0$.
    until $\mathrm{Rem} = 0$ *or* $\{i : \mathrm{Rem}(i) > 0\} = \emptyset$; **end**

Algorithm 4 (Online Proportional)

Input: Sequence of loads $(L_t)_{t=1}^T$ and SLAs $\beta(1),\dots,\beta(N)$.

1. Initial distribution $\mathbf{h}_1 = (\beta(1),\dots,\beta(N))$,
2. **For** $t = 1,\dots,T$, **do**
3. Set $\mathbf{h}_t$ and obtain $A_t = \{i : Q_t(i) \neq 0\}$,
4. Update $h_{t+1}(i) = \frac{\beta(i)}{\sum_{j\in A}\beta(j)}$
5. If $A = \emptyset$, then $\mathbf{h}_{t+1} = \mathbf{h}_1$.

Appendix F: Gamma Distribution

Recall that a gamma distribution (Feller 1957) is characterized by two parameters: the shape $k > 0$ and the scale $\theta > 0$. The PDF of a gamma(k,θ) is given by $\frac{1}{\Gamma(k)\theta^k}x^{k-1}e^{-x/\theta}$, where $\Gamma(k) = \int_0^\infty u^{k-1}e^{-u}du$ is the standard gamma function. See Figure 8 for PDFs of different gamma distribution for various choices of k and θ.

Proposition 3. *Let* $X \sim \mathrm{Gamma}(k,\theta)$. *Then,* $\mathbb{E}[X] = k\theta$ *and* $\mathrm{Var}(X) = k\theta^2$.

References

Abernethy J, Bartlett PL, Rakhlin A, Tewari A (2008) Optimal strategies and minimax lower bounds for online convex games. Accessed May 1, 2018, https://repository.upenn.edu/statistics_papers/164.

Albers S (2003) Online algorithms: a survey. *Math. Programming* 97(1-2):3–26.

Arora S, Hazan E, Kale S (2012) The multiplicative weights update method: a meta-algorithm and applications. *Theory Comput.* 8(1):121–164.

Ben-Tal A, Nemirovski A (2001) *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, vol. 2 (Siam, Philadelphia).

Borodin A, El-Yaniv R (2005) *Online Computation and Competitive Analysis* (Cambridge University Press, Cambridge, UK).

Boyd S, Vandenberghe L (2004) *Convex Optimization* (Cambridge University Press, Cambridge, UK).

Bubeck S, Cesa-Bianchi N, et al (2012) Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations Trends® in Machine Learn.* 5(1):1–122.

Buchbinder N, Naor JS (2009) *The Design of Competitive Online Algorithms via a Primal-Dual Approach* (Now Publishers, Inc., Hanover, MA).

Cohen MC, Keller PW, Mirrokni V, Zadimoghaddam M (2019) Overcommitment in cloud services: Bin packing with chance constraints. *Management Sci.* 65(7):2947–3448.

Curino C, Difallah DE, Douglas C, Krishnan S, Ramakrishnan R, Rao S (2014) Reservation-based scheduling: If you're late don't blame us! *Proc. ACM Symposium Cloud Comput.* (ACM), 1–14.

Dabbagh M, Hamdaoui B, Guizani M, Rayes A (2015) Efficient datacenter resource utilization through cloud resource overcommitment. *2015 IEEE Conf. Comput. Comm. Workshops (INFOCOM WKSHPS)* (IEEE), 330–335.

Feller W (1957) *An Introduction to Probability Theory and Its Applications* (John Wiley & Sons, New York).

Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.* 55(1):119–139.

Gera A, Xia CH (2011) Learning curves and stochastic models for pricing and provisioning cloud computing services. *Service Sci.* 3(1):99–109.

Ghodsi A, Zaharia M, Hindman B, Konwinski A, Shenker S, Stoica I (2011) Dominant resource fairness: Fair allocation of multiple resource types. *NSDI* 11:24.

Gordon A, Hines M, Da Silva D, Ben-Yehuda M, Silva M, Lizarraga G (2011) Ginkgo: Automated, application-driven memory overcommitment for cloud computing. *ASPLOS RESoLVE Workshop*, 1–6.

Grandl R, Chowdhury M, Akella A, Ananthanarayanan G (2016) Altruistic scheduling in multi-resource clusters. *OSDI*, 65–80.

Hall EC, Willett RM (2015) Online convex optimization in dynamic environments. *IEEE J. Sel. Top. Signal Process.* 9(4):647–662.

Hazan E (2019) Introduction to online convex optimization. Preprint, submitted September 7, https://arxiv.org/abs/1909.05207

Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz RH, Shenker S, Stoica I (2011) Mesos: A platform for fine-grained resource sharing in the data center. *NSDI* 11:22.

Jyothi SA, Curino C, Menache I, Narayanamurthy SM, Tumanov A, Yaniv J, Mavlyutov R, et al. (2016) Morpheus: Toward automated slos for enterprise clusters. *OSDI*, 117–134.

Li Cp, Neely MJ (2009) Energy-optimal scheduling with dynamic channel acquisition in wireless downlinks. *IEEE Trans. Mobile Comput.* 9(4):527–539.

Macías M, Guitart J (2011) A genetic model for pricing in cloud computing markets. *Proc. 2011 ACM Symposium Appl. Comput.* (ACM), 113–118.

Maguluri ST, Srikant R (2014) Scheduling jobs with unknown duration in clouds. *IEEE/ACM Trans. Networking* 22(6):1938–1951.

Maguluri ST, Srikant R, Ying L (2012) Stochastic models of load balancing and scheduling in cloud computing clusters. *Proc. IEEE INFOCOM* (IEEE), 702–710.

Maguluri ST, Srikant R, Ying L (2014) Heavy traffic optimal resource allocation algorithms for cloud computing clusters. *Performance Evaluation* 81:20–39.

Menache I, Singh M (2015) Online caching with convex costs. *Proc. 27th ACM Symposium Parallelism Algorithms Architectures* (ACM), 46–54.

Mohri M, Rostamizadeh A, Talwalkar A (2018) *Foundations of Machine Learning* (MIT Press, Cambridge, MA).

Mokhtari A, Shahrampour S, Jadbabaie A, Ribeiro A (2016) Online optimiziation in dynamic environments: improved regret rates for strongly convex problems. *Decision Control (CDC), 2016 IEEE 55th Conf.*, (IEEE) 7195–7201.

Narasayya V, Das S, Syamala M, Chandramouli B, Chaudhuri S (2013) SQLVM: performance isolation in multi-tenant relational database-as-a-service. *6th Biennial Conf. Innovative Data Systems Res.*

Narasayya V, Menache I, Singh M, Li F, Syamala M, Chaudhuri S (2015) Sharing buffer pool memory in multi-tenant relational database-as-a-service. *Proc. VLDB Endowment* 8(7):726–737.

Neely MJ (2007) Optimal energy and delay tradeoffs for multiuser wireless downlinks. *IEEE Trans. Inform. Theory* 53(9):3095–3113.

Neely MJ (2008) Order optimal delay for opportunistic scheduling in multi-user wireless uplinks and downlinks. *IEEE/ACM Trans. Networking* 16(5):1188–1199.

Passacantando M, Ardagna D, Savi A (2016) Service provisioning problem in cloud and multi-cloud systems. *INFORMS J. Comput.* 28(2):265–277.

Plotkin SA, Shmoys DB, Tardos É (1995) Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.* 20(2):257–301.

Rasley J, Karanasos K, Kandula S, Fonseca R, Vojnovic M, Rao S (2016) Efficient queue management for cluster scheduling. *Proc. Eleventh Eur. Conf. Comput. Systems* (ACM), 36.

Shalev-Shwartz S (2012) Online learning and online convex optimization. *Foundations Trends® Machine Learn.* 4(2): 107–194.

Sharma B, Thulasiram RK, Thulasiraman P, Garg SK, Buyya R (2012) Pricing cloud compute commodities: A novel financial economic model. *Proc. 2012 12th IEEE/ACM Internat. Symposium Cluster, Cloud Grid Comput. (ccgrid 2012)* (IEEE Computer Society), 451–457.

Shirani-Mehr H, Caire G, Neely MJ (2010) Mimo downlink scheduling with non-perfect channel state knowledge. *IEEE Trans. Comm.* 58(7):2055–2066.

Srikant R, Ying L (2013) *Communication networks: an optimization, control, and stochastic networks perspective* (Cambridge University Press, Cambridge, MA).

Tassiulas L, Ephremides A (1990) Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *29th IEEE Conf. Decision Control* (IEEE), 2130–2132.

Tassiulas L, Ephremides A (1993) Dynamic server allocation to parallel queues with randomly varying connectivity. *IEEE Trans. Inform. Theory* 39(2):466–478.

Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, et al (2013) Apache hadoop yarn: Yet another resource negotiator. *Proc. 4th Annual Symposium Cloud Comput.* (ACM), 5.

Zaharia M, Borthakur D, Sen Sarma J, Elmeleegy K, Shenker S, Stoica I (2010) Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. *Proc. 5th Eur. Conf. Comput. Systems* (ACM), 265–278.

Zhang L, Yang T, Yi J, Rong J, Zhou ZH (2017) Improved dynamic regret for non-degenerate functions. Neural Information Processing Systems.

Zinkevich M (2003) Online convex programming and generalized infinitesimal gradient ascent. *Proc. 20th Internat. Conf. Machine Learn. (ICML-03)*, 928–936.

Sebastian Perez-Salazar is a PhD student in the H. Milton Stewart School of Industrial & Systems Engineering (ISyE) at Georgia Institute of Technology. His research interests lie in the intersection of optimization under uncertainty and dynamic resource allocation. His research has focused on optimization problems in cloud computing, online advertising, and scheduling problems.

Ishai Menache got his PhD in Electrical Engineering from the Technion, Israel. He was a postdoctoral associate at the Laboratory for Information and Decision Systems at MIT. Ishai has been with Microsoft Research since 2011, where he is the founder and manager of the Cloud Operations Research (CORE) group. His research focuses on developing large-scale optimization frameworks for cloud systems and applications. He is also interested in systems and networking, optimization, and machine learning.

Mohit Singh is an associate professor in the H. Milton Stewart School of Industrial & Systems Engineering (ISyE) at Georgia Institute of Technology. Singh's research interests include discrete optimization, approximation algorithms, and convex optimization. His research has focused on optimization problems arising in cloud computing, logistics, network design, and machine learning.

Alejandro Toriello is an associate professor in the H. Milton Stewart School of Industrial & Systems Engineering (ISyE) at Georgia Institute of Technology. He currently holds the Benatar Early Career Professorship. His research interests lie in discrete and dynamic optimization and its applications, especially in logistics, transportation, cloud computing, and online advertising.