Modeling and Analyzing Waiting Policies for Cloud-Enabled Schedulers

Pradeep Ambati, Noman Bashir, David Irwin, and Prashant Shenoy
University of Massachusetts Amherst

Abstract—Cloud platforms have popularized the Infrastructure-as-a-Service (IaaS) purchasing model, which enables users to rent computing resources on demand to execute their jobs. However, buying fixed resources is still much cheaper than renting if their resource utilization is high. Thus, to optimize cost, users must decide how many fixed resources to provision versus rent "on demand" based on their workload. In this paper, we introduce the concept of a *waiting policy* for cloud-enabled schedulers and show that the optimal cost depends on it. The waiting policy explicitly controls how long jobs wait for resources, as jobs never need to wait, since cloud platforms provide the illusion of infinite scalability. A waiting policy is the dual of a scheduling policy: while a scheduling policy determines which jobs should run when fixed resources are available, a waiting policy determines which jobs should wait when fixed resources are not available. We define multiple waiting policies and develop simple and general analytical models to reveal their tradeoff between fixed resource provisioning, cost, and job waiting time. We evaluate the impact of different waiting policies on a real year-long batch workload consisting of 14M jobs run on a 14.3k-core cluster. We show that a compound waiting policy, which forces jobs with long running times or short waiting times to wait for fixed resources, offers the best tradeoff. The policy decreases both the cost (by 5%) and mean job waiting time (by 7$\times$) compared to the current cluster, and also decreases the cost (by 43%) compared to renting on-demand resources for a modest increase in mean job waiting time (at 1.74 hours).

1 Introduction

Cloud platforms have popularized the Infrastructure-as-a-Service (IaaS) purchasing model by enabling users to programmatically rent computing resources on demand, in the form of virtual machines (VMs), to execute their jobs.[1] Many large enterprises are now partially or completely migrating their private computing infrastructure to cloud platforms. For example, Netflix shut down its last private data center in 2016 after entirely migrating its video streaming service to Amazon Web Services (AWS) [13]. Cloud-enabled infrastructure uses similar software systems as private clusters to manage resources at large scales, typically consisting of a centralized job scheduler, such as Slurm [4] or Kubernetes [2]. Users submit jobs, with specified resource requirements, to these schedulers, which either allocate idle resources to execute them or force them to wait for idle resources to become available. Since private clusters manage a *fixed* number of computing resources typically sized for peak demands, they often have low average utilization (<30%), but may periodically experience large bursts in job arrivals, e.g., due to deadlines, product releases, or seasonal variations, that result in long job waiting times.

As job schedulers migrate to the cloud, they have many options for *optimizing cost* and *reducing job waiting times*. For example, schedulers may provision cloud VMs on demand to service jobs, requiring them to only pay for resources when jobs need them. In this case, the cloud's operating costs are often much lower than the capital cost of an under-utilized fixed-size cluster, since the latter must effectively "pay" when resources are idle. In addition, since the cloud provides the illusion of infinite scalability, jobs never need to wait for resources, as schedulers can always acquire cloud resources to service them immediately. Most schedulers are now cloud-enabled and support such "auto-scaling," which acquires cloud VMs to service jobs, and releases them when done [1], [3].

Importantly, however, *buying fixed resources (or reserving them for long periods) is significantly cheaper than renting resources on demand if the fixed resources are highly utilized.* Cloud pricing models make this clear, as reserving a VM for 1-3 years costs 40-60% less per-hour than renting an equivalent on-demand VM over the same period. For example, reserving a `m5.large` VM from AWS, which includes 2 cores and 8GB RAM, for 3 years currently costs $988, while renting it on demand costs $0.096/hour or $2,522.88 over the same period. Of course, fixed resources are only cost-effective if they are highly utilized: if jobs only execute on the `m5.large` for less than a third of the time, the on-demand option is cheaper (at a cost of $840.96). The cost advantage of buying versus renting is even greater for specialized hardware with a recent analysis estimating that purchasing a GPU-based deep learning cluster costs 90% less than renting one on demand from AWS [14]. Thus, a mixed infrastructure that satisfies some baseload with highly-utilized fixed resources, and satisfies load bursts using on-demand resources can decrease cost. Notably, hybrid clouds, which combine fixed private resources with cloud bursting, use this approach [5], [21], [28], as do many companies, which both buy reserved VMs and dynamically rent on-demand VMs [22]. As we discuss in §7, physical infrastructure is also becoming networked and programmatically driven, which has the potential to spread the cloud model to other sectors, such as transportation and energy, where schedulers may choose between *buying* fixed resources or *renting* them to service various "jobs."

To address the problem, in this paper, we introduce the concept of a *waiting policy* for cloud-enabled schedulers, and show that provisioning fixed resources to optimize cost is dependent on it. The waiting policy explicitly controls whether and how long jobs wait for fixed resources before deciding to run them on on-demand resources. A waiting policy is the dual of a scheduling policy:

1. This is an expanded and revised version of a preliminary paper that appeared at Supercomputing 2020 [12].

while a scheduling policy determines which jobs run when fixed resources are available, a waiting policy determines which jobs wait for fixed resources when they are not available (rather than run immediately on on-demand resources). While there has been decades of work on job scheduling policies, we know of no prior work that defines or analyzes waiting policies, which are distinct from scheduling policies in that cloud-enabled schedulers define both independently of each other. For cloud-enabled schedulers, the waiting policy is just as important as the scheduling policy, since it dictates the tradeoff between job performance and cost. Waiting policies also differ from auto-scaling policies currently used by cloud-enabled schedulers, which immediately acquire resources to satisfy queued jobs without any waiting [6].

Clearly, the longer jobs are willing to wait for fixed resources, the higher their utilization, and the lower their overall cost. However, as we show, the relationship and tradeoff between the number of fixed resources, the waiting policy, and the optimal cost is non-intuitive. *To better understand these tradeoffs*, we define multiple fundamental non-selective and selective waiting policies and develop simple analytical queueing models for them. Non-selective waiting policies apply the same policy to all jobs, while selective waiting policies apply the policy to only selected jobs based on system or job characteristics. While we focus on waiting policies for cloud-enabled job schedulers, such as Slurm [4] and Kubernetes [2], our analytical models are general and thus may also be applicable to schedulers for other resources, as we discuss in §7. As we show, while these analytical models are not predictive, since their scheduling policy and workload assumptions do not always hold in practice, they enable users to better understand and reason about the impact of waiting policies by understanding how a system's characteristics differ from the models' assumptions. Our hypothesis is that, by optimizing their waiting policy, cloud-enabled schedulers can reduce job waiting times, while mitigating the impact on cost, or vice versa. In evaluating our hypothesis, we make the following contributions.

Introduce a Waiting Policy. We introduce the concept of a waiting policy for cloud-enabled schedulers, and present multiple fundamental non-selective and selective waiting policies. Our non-selective waiting policies include All Jobs Wait (AJW), No Jobs Wait (NJW), and All Jobs Wait Threshold (AJW-T), while our selective policies include Short Waits Wait (SWW) and Long Jobs Wait (LJW). Since waiting policies are not mutually exclusive, we also present a compound policy that concurrently applies AJW-T, SWW, and LJW to gain the benefits of all three.

Waiting Policy Models and Analysis. We show how to analyze waiting policies for cloud-enabled schedulers in general using a simple queuing model to understand their tradeoff between fixed resource provisioning, cost, and job waiting time. Our approach extends classic marginal analysis by combining it with a number of different queuing results and analyses to model cost under job waiting. We then apply this approach to model, analyze, and empirically validate each waiting policy above to demonstrate the importance of explicitly defining a waiting policy to optimize cost for cloud-enabled schedulers. Our modeling and analysis also provides the necessary formal foundation for conducting any future work on waiting policies for cloud-enabled schedulers.

Modeling and Analysis Under Uncertainty. As with many scheduling policies, our waiting policies require *a priori* knowledge of job running times and waiting times, which is not always available. Since predictions of job running times and waiting times may be inaccurate, we extend our models and analyses above to quantify the effect of inaccurate predictions on our waiting policies. Our analysis reveals an interesting asymmetry in that our waiting policies are highly sensitive to over-predictions of waiting time, but not to under-predictions.

Purchasing Option (utilization%)	Raw Price	Effective Price	3-year Cost	Normalized Price
On-demand (100%)	9.6¢/hr	9.6¢/hr	$2523	~ 1.0
On-demand (60%)	9.6¢/hr	9.6¢/hr	$1514	~ 1.0
On-demand (40%)	9.6¢/hr	**9.6¢/hr**	**$1009**	**$\sim$1.0**
Fixed Reserved (100%)	3.8¢/hr	3.8¢/hr	$988	~ 0.4
Fixed Reserved (60%)	3.8¢/hr	6.3¢/hr	$988	~ 0.7
Fixed Reserved (40%)	3.8¢/hr	**9.5¢/hr**	**$988**	**$\sim$1.0**

TABLE 1
Raw price, effective price per unit time of utilized resources, 3-year cost, and normalized price for different utilizations of a fixed reserved and on-demand VM from AWS.

Implementation and Evaluation. We implement our waiting policies in a trace-driven job simulator, and evaluate their impact on a real year-long batch computing workload consisting of 14 million (M) jobs run on a 14k-core cluster. The results show that our compound policy offers the best tradeoff: it decreases the cost (by 5%) and mean job waiting time (by $7\times$) compared to the current cluster using AJW, and decreases the cost (by 43%) compared to only renting on-demand resources for a modest increase in mean job waiting time (at 1.74 hours).

2 Background and Intuition

We provide background on cloud pricing of fixed and on-demand VMs, and applying marginal analysis to optimize cost.

Pricing Dynamics. We focus on applying waiting policies using the pricing dynamics of existing cloud platforms. As we discuss in §7, these pricing dynamics are both fundamental and general, and thus may apply to other resources where a similar buy versus rent option is available. We assume a cloud platform that offers two types of resources: on-demand and fixed. Users may acquire and release on-demand resources any time, and pay only for the time they use them without any commitment. In contrast, users must commit to paying for fixed resources over a long period, e.g., one or more years. Importantly, however, fixed resources are cheaper than on-demand resources if they are highly utilized.

Table 1 shows the pricing dynamics of an on-demand and fixed (3-year reserved) `m5.large` cloud VM on AWS in the U.S. East region. The table includes the raw price per unit time, the effective price of utilized resources, 3-year cost, and normalized price, i.e., the effective price relative to the raw on-demand price, for each scenario. As mentioned in §1, the on-demand VM's 3-year cost is much higher than the fixed VM's cost at 100% utilization. However, the fixed VM's cost is constant and independent of its utilization due to the long-term commitment, while the on-demand VM's cost changes with utilization, since users release it when not in use. Here, utilization simply denotes the fraction of non-idle periods over time. Since the fixed VM's resources are wasted during idle periods, its *effective price* for utilized resources increases with decreasing utilization. In this case, if the fixed VM is utilized $>$40% of the time, its effective price and 3-year cost are less than the on-demand VM, thereby making it the cheaper option. We call this the *break even point*.

The cost dynamics above are fundamental to the economics of any platform that rents resources, since the platform must always recoup its own costs for buying fixed resources, in addition to any operating costs and profit, by renting them to users. By serving a large pool of users with different resource requirements, these

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems

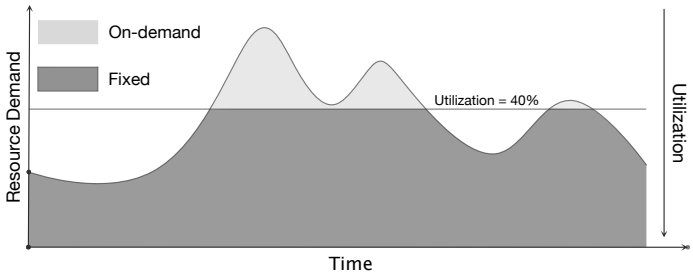


Fig. 1. Illustration of utilization for each unit of stacked resource demand and the break even point at 40% utilization.

platforms are able to operate their fixed resources at a much higher resource utilization than any single user, which results in a much lower effective price. Volume discounts and higher operational efficiency at large scales, i.e., "economies of scale," can also contribute to lowering these platforms' effective price for fixed resources. Even so, as our example illustrates, highly utilized fixed resources are still much cheaper, since they eliminate the platform's primary cost advantage.

Marginal Analysis. In economics, marginal analysis examines the additional benefits of some activity compared to the additional costs incurred by that activity. Determining the optimal mix of fixed and on-demand resources to execute a workload on a cloud platform to minimize cost is a classic marginal analysis problem [25]. Given a workload and some fixed resources capable of servicing a fraction of it, the marginal analysis problem is to determine whether the additional benefit of acquiring one more fixed resource to serve (a portion of) the remaining workload outweighs its cost, i.e., the savings from renting an on-demand resource to service the same portion.

Figure 1 illustrates marginal analysis pictorially for an example workload where time is on the x-axis and resource demand is on the y-axis. We assume the fixed and on-demand resources have the same prices as in Table 1. To determine the optimal mix of fixed and on-demand resources using marginal analysis, we simply add fixed resources, one at a time, to satisfy each unit of stacked resource demand in order (starting from 0 on the y-axis) up to the point where the utilization of the fixed resource equals our break even point on the y-axis, which is 40% (in dark grey). When the instantaneous demand exceeds the fixed resource capacity at the horizontal line (in light grey), dynamically acquiring and releasing on-demand resources to satisfy the remaining workload is cheaper.

More formally, let p_f and p_o denote the price per unit time for a fixed resource (at 100% utilization) and on-demand resource, respectively, let d denote the discount factor for a fixed resource, such that $p_f{=}d\times p_o$ and $0{\leq}d{\leq}1$, and let T denote the workload's duration. The cost of adding one more fixed resource s over the workload's duration T is $p_f \times T$. Now suppose this s^{th} resource operates at utilization ρ_s when servicing the remaining workload. Since the scheduler can acquire and release on-demand resources at any time, the cost of servicing the remaining workload using an on-demand resource is $\rho_s \times T \times p_o$, as the scheduler can acquire the on-demand resource in $\rho_s \times T$ time slots and release it when idle. Thus, using a fixed resource is only cheaper if $p_f \times T < \rho_s \times T \times p_o$. By substituting $p_f{=}d \times p_o$, we observe that only when $d{<}\rho_s$, or the discount factor is less than the utilization of the last fixed resource we added, is acquiring an additional fixed resource cheaper than using on-demand resources. Similarly, the cost of provisioning an additional fixed or on-demand resource is equal when $\rho_s{=}d$, or the discount factor equals the utilization of the last fixed resource. Beyond this break even point, there is no marginal cost savings from acquiring more fixed resources.

The marginal analysis problem above is straightforward to solve in the context of a traditional queuing model using classic results by Erlang, assuming arriving jobs never wait for resources [19], [31], [35]. Variants of this classic problem have been addressed in prior work both generally, and in the context of cloud computing, which we discuss in §8.

Marginal Analysis under Waiting. The classic marginal analysis above implicitly assumes jobs never wait for resources, and always immediately execute on either a fixed or on-demand resource. A key insight of our work is that cloud-enabled schedulers can explicitly control whether (and how long) jobs wait for fixed resources if they are busy, and that this waiting policy affects the optimal provisioning of fixed resources that minimizes cost. In general, the longer the permissible waiting time, the higher the fixed resource utilization, and the lower the overall cost. As we show, cloud-enabled schedulers can implement a wide variety of waiting policies that offer different tradeoffs between fixed resource provisioning, cost, and job waiting time. Despite the importance of the waiting policy in optimizing cost when using cloud platforms, we know of no work that explicitly defines and analyzes such waiting policies for cloud-enabled schedulers by applying marginal analysis.

3 Non-selective Waiting Policies

We develop a simple queuing model for cloud-enabled schedulers to understand the relationship between the waiting policy, fixed resource provisioning, job waiting time, and cost. While our implementation and evaluation in §5 and §6 focus on cloud platforms, our queuing model and analysis are general and may apply to similar schedulers for other resources, as we discuss in §7. We first analyze basic non-selective waiting policies—All Jobs Wait (AJW), No Jobs Wait (NJW), and All Jobs Wait Threshold (AJW-T)—which apply the same policy to all jobs. In §4, we analyze selective waiting policies that only force selected jobs to wait based on their characteristics.

Our analysis extends a $M/M/s/\infty$ queuing model using s fixed resources with first-come-first-serve (FCFS) scheduling, mean job arrival rate λ, and mean job service time $1/\mu$, where job arrivals follow a Poisson process, job service times are independent and identically distributed (i.i.d.). and exponentially distributed, and each resource executes one job at a time. The offered load is $a{=}\lambda/\mu$, and the offered load (and mean utilization) per fixed resource is $\rho{=}a/s{=}\lambda/(s{\times}\mu)$. Our analysis only applies in steady-state. For reference, Table 2 lists each variable our analysis uses. We use a standard queuing model because it permits a closed-form analysis to understand the basic tradeoffs in designing waiting policies. We then show how better understanding these tradeoffs can enable users to reason about the effects of waiting policies on systems that do not conform to this model, e.g., by having different scheduling policy or workload characteristics.

3.1 All Jobs Wait

Model Analysis. All Jobs Wait (AJW) is a baseline policy that requires all jobs to wait for fixed resources, and never rents on-demand resources. We present it as a foundation for our subsequent analysis. AJW's analysis is equivalent to that of an $M/M/s/\infty$ queue. The effective price P for each fixed resource

Variable	Description	Units
s	Number of Resources	-
λ	Mean Job Arrival Rate	jobs/time
μ	Mean Service Rate	time/job
a	Offered Load - λ/μ	%
ρ	Fixed Resource Utilization - $a/s = \lambda/(s \times \mu)$	%
P	Amortized Price of Utilized Resources	$/time
p_f	Fixed Resource Price at 100% Utilization	$/time
$C(s,a)$	Erlang's C (or delay) formula	time
w	Mean Wait Time	time
p_o	On-demand Resource Price	$/time
d	Discount Factor - p_f/p_o	%
$B(s,a)$	Erlang's B (or loss) formula	%
ρ_s	Utilization of s^{th} Resource	%
r	Fraction of Jobs using On-demand Resources	%
T	Workload Duration	time
C	Workload Cost over T	$
S	Cost Savings versus On-demand Resources	$
b	Maximum Waiting Time Threshold	time
r_s	r from above when using s fixed resources	%
r_{s-1}	r from above when using $s-1$ fixed resources	%
f_{under}	Fraction of Jobs Under-predicting Wait or Run Time	%
f_{over}	Fraction of Jobs Over-predicting Wait or Run Time	%
μ_{long}	Mean Service Rate of Long Jobs in LJW	time/job
μ_{short}	Mean Service Rate of Short Jobs in LJW	time/job
P_{long}	Amortized price of long jobs in LJW	$/time
P_{short}	Amortized price of short jobs in LJW	$/time
w_{long}	Waiting time of long jobs in LJW	time
t	Long/short Job Threshold	time
λ_{long}	Mean arrival rate of long jobs	jobs/time
ρ_{long}	Fixed resource utilization when running long jobs	%
r_{short}	Fraction of short jobs	%
r_{sww}	Fraction of long jobs with short waits	%

TABLE 2
Listing of symbol, description, and units for each variable we use in our analysis roughly in order of introduction.

is simply a function of the mean resource utilization ρ and fixed resource price p_f at full utilization, as shown below.

$$P = p_f/\rho \tag{1}$$

Thus, as mean utilization ρ increases, the effective price decreases up to 100% utilization. Of course, as utilization increases, the mean waiting time w in the queue also increases. The mean waiting time w for fixed resources under AJW is a well-known function, shown below, of s, λ, and μ, where $C(s,a)=[(s \times a^s)/(s! \times (s-a))]/[\sum_{i=0}^{s-1} a^i/i! + (s \times a^s)/(s! \times (s-a))]$ is Erlang's delay (or C) formula.

$$w = \frac{C(s,a)}{s \times \mu - \lambda} \tag{2}$$

Empirical Validation. *We empirically validate the effective price P and mean waiting time w for all models we present in §3 and §4 for the same baseline example.* In our baseline example, we set λ=0.2 (or 1 job every 5 seconds on average), μ=0.002 (or an average job service time of 500 seconds), p_o=9.6¢/hour, and p_f=3.84¢/hour. Thus, in this case, the discount factor d for fixed resources at 100% utilization is p_f/p_o=0.4. As in our example in §2, we set p_o and p_f based on the on-demand and 3-year reserved VM prices in AWS, and set λ and μ such that the mean utilization ρ of the fixed resources is 100% when s=100 resources. We plot both the continuous function from our model, as well as average empirical values from 20 trials of our job simulator from §5. Each trial simulates the model on a synthetically generated job trace with 2 million jobs using exponentially distributed inter-arrival and service times based on the baseline parameters, as well as any model-specific parameters. To capture steady states, we do not include the first and last 10% of jobs when computing P and w. All graphs include error bars representing the maximum and minimum across all trials, although, with 2 million jobs, there is almost no deviation from the average on each trial.

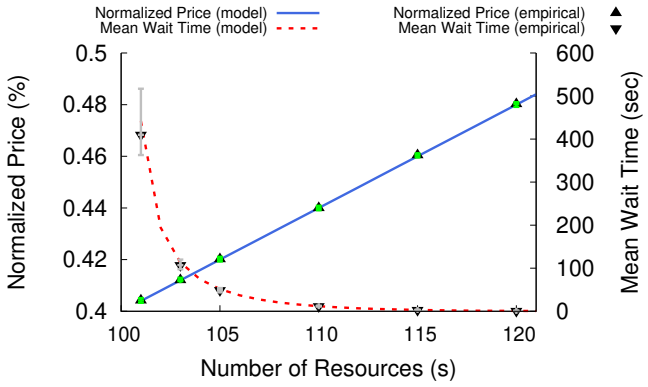


Fig. 2. Normalized price P (left y-axis) and mean wait time w (right y-axis) as a function of fixed resources s under AJW. Mean wait time $w \to \infty$ as fixed resources $s \to 100$, and mean wait time $w \to 0$ as fixed resources $s \to \infty$.

For AJW, Figure 2 plots the effective price P (left y-axis), obtained from our model and from simulations, as a function of the fixed resources s. Here, *as in all subsequent graphs*, we normalize the effective price P by the price of on-demand resources p_o. Thus, the left y-axis represents how much using fixed resources lowers or raises the price relative to using on-demand resources; smaller numbers (lower prices) are better. The minimum value on the left y-axis is P=p_f=0.4, since this represents the lowest possible price (using only fixed resources at 100% utilization). The right y-axis shows the mean waiting time w for fixed resources.

Figure 2 shows that our model's predictions closely match the empirical results, both for the normalized price and the mean waiting time. Also, as expected, the graph shows that as s increases the effective price P increases linearly due to the decrease in mean utilization ρ. In contrast, the mean waiting time decreases super-linearly with increasing s. Thus, AJW offers a risky tradeoff between w and P, since provisioning fixed resources for high utilization, i.e., a low s, to reduce the price may cause high waiting times. As a result, AJW encourages over-provisioning to ensure waiting times near 0 that are outside the region where they increase super-linearly.

The effective price P equals the on-demand price p_o when the mean utilization of fixed resources ρ equals the discount factor d=0.4, which occurs at s=250 (not shown). Thus, provisioning any fixed resources s<250 is cheaper than solely using on-demand resources. Reducing s to 120 still yields a waiting time $w \sim 0$ for an effective price P that is 52% lower than s=250 and only 20% higher than s=100 where $w \to \infty$.

Key Point. *Since waiting time increases super-linearly as utilization $\rho \to 100\%$, AJW encourages over-provisioning to ensure a utilization below 100% with waiting times near* 0.

3.2 No Jobs Wait

Model Analysis. The No Jobs Wait (NJW) waiting policy is similar to existing auto-scaling policies for cloud-enabled schedulers that execute jobs on fixed resources when available, and dynamically acquire on-demand resources to execute jobs when all fixed resources are busy. Recall from §2 that, given a workload, there is an optimal number of fixed resources s for NJW that minimizes cost, and this value occurs when the s^{th} resource has a utilization equal to the fixed resource's discount factor d. Thus, to optimize s under NJW, we need an expression for the s^{th} resource's utilization, denoted as ρ_s.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems

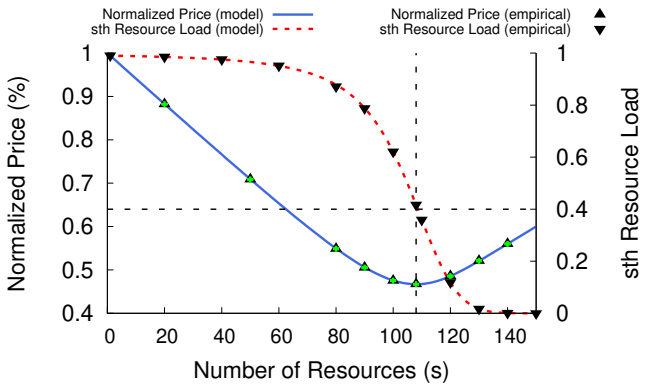


Fig. 3. Normalized price P (left y-axis) and mean utilization of the sth resource ρ_s (right y-axis) as a function of fixed resources s under NJW. Minimum price occurs when the fixed resources' discount factor d=ρ_s.

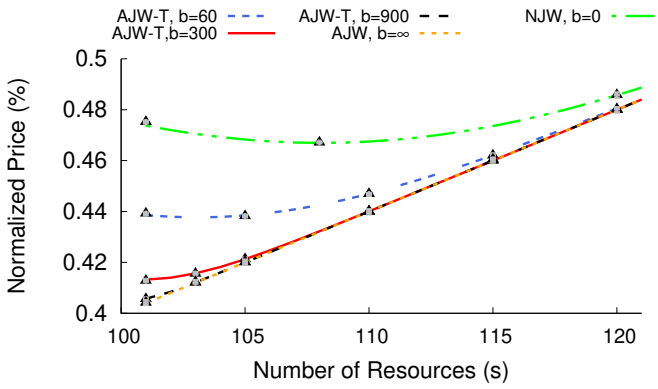


Fig. 4. Normalized price P as a function of fixed resources s under AJW-T for different threshold waiting times b.

We find ρ_s using marginal analysis by applying Erlang's loss (or B) formula, which assumes a $M/M/s/0$ queue. Since the queue size is zero, any job that arrives and observes all resources as busy must exit the system. Erlang's loss formula gives the blocking probability that an arriving job exits the system, or equivalently that there are s jobs in the system and all resources are busy. To compute the utilization of the sth resource, we first compute the difference between the blocking probability when using $s-1$ and when using s resources. This difference represents the percentage of jobs an additional resource serves. Multiplying this percentage by the offered load a=λ/μ gives the mean utilization of the sth resource ρ_s, as shown below, where $B(s,a)$=$(a^s/s!)/(\sum_{i=0}^{s}(a^i/i!))$ is Erlang's loss (or B) formula.

$$\rho_s = a \times [B(s-1,a) - B(s,a)] \tag{3}$$

Under a No Jobs Wait (NJW) waiting policy, rather than actually exit the system, the scheduler acquires on-demand resources to immediately service blocking jobs without waiting. To determine the optimal number of fixed resources s that minimizes cost, we set the discount factor d equal to ρ_s in Equation (3) and solve for s. Since Erlang's loss formula includes a factorial and summation, there is no closed-form expression for s, requiring us to solve for it numerically. Since ρ_s is monotonically decreasing as s increases, we can use a binary search to determine the optimal s. After solving for s, we compute the minimum effective price P per resource per unit time for the s fixed resources and additional on-demand resources necessary to satisfy the offered load.

$$P = (1-r) \times \frac{p_f}{\rho_f} + r \times p_o \tag{4}$$

Here, we use r to represent the fraction of the workload that executes on on-demand resources. The first additive term normalizes the price of the s fixed resources p_f at 100% utilization by their mean utilization ρ_f, which is $(1-r) \times \rho$, since the mean arrival rate to the s fixed resources is only $(1-r) \times \lambda$. We then multiply this normalized price by the fraction of load $(1-r)$ serviced at this price. The second additive term simply multiplies the price of on-demand resources p_o by the remaining fraction of the workload r. For NJW, r=$B(s,a)$, as this represents the probability that a job blocks and then runs on on-demand resources. Since jobs block uniformly at random, the mean service time of blocking and non-blocking jobs both equal the mean service time $1/\mu$. As a result, we need not weight each additive term in Equation (4) by its fraction of the mean service time.

The total cost C (in dollars) to execute a workload over time T, i.e., the fixed resources' lifetime, is then shown below.

$$C = P \times (\frac{1}{\mu}) \times (\lambda T) = s \times p_f \times T + r \times \frac{\lambda}{\mu} \times p_o \times T \tag{5}$$

The total cost C is the product of the effective price per unit time P, the mean service time per job ($1/\mu$), and the total number of jobs, which in-turn is the product of the job arrival rate λ and the total time T. We can also represent the total cost in a different, but equivalent, way on the right side by expanding P using Equation (4). Here, the first additive term is the cost for the s fixed resources over time T, and the second term is the cost of renting on-demand resources. The first term is independent of the offered load, since users must pay for the s fixed resources regardless of their utilization. Of course, Equation (5) for C only applies to the system in steady state over the interval T. As noted earlier, our simulations capture steady state by not considering the first and last 10% of jobs when computing C, P, or w.

Empirical Validation. We empirically validate NJW using the same baseline example from §3.1. Figure 3 shows the effective price P (left y-axis) as a function of fixed resources s under NJW, where we again normalize P by the price of on-demand resources p_o. The right y-axis shows the mean utilization of the sth resource ρ_s, as the waiting time w is always zero under NJW. As expected, the graph shows the model closely matches the empirical results. As s increases, the effective price decreases to the optimal s=108 where ρ_s equals the 0.4 discount factor, after which, the effective price increases. Plugging the optimal s value and our baseline parameters into Equation (3) verifies that ρ_s=0.4.

At the optimal s=108, NJW has an effective price P=0.467×0.096=\$0.044832/hour, while AJW's price is $\sim$7.5% less at P=0.432×0.096=\$0.041472/hour. However, under NJW, jobs never incur waiting time, while AJW incurs a mean waiting time of 20s, with some jobs waiting much longer. Thus, for 7.5% higher cost, NJW guarantees jobs never wait. In this case, r=0.035, i.e., 3.5% of jobs run on on-demand resources, which results in a minimum cost (in dollars) over a 3-year period of C=\$117,818. By contrast, solely using on-demand resources costs $100(0.096)(26280)$=\$252,288, which is over twice as expensive as the optimal cost under NJW.

Key Point. *While NJW's cost is higher than AJW's for the same fixed resources, it guarantees no waiting time. NJW encourages optimal provisioning, since its cost increases as fixed resource provisioning deviates from the optimal.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems

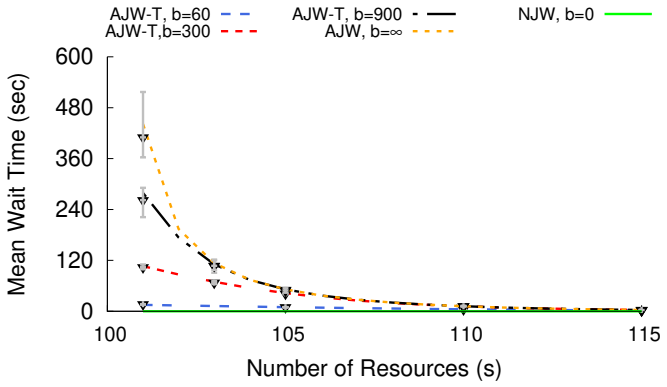


Fig. 5. Mean waiting time w as a function of fixed resources s under AJW-T for different threshold waiting times b.

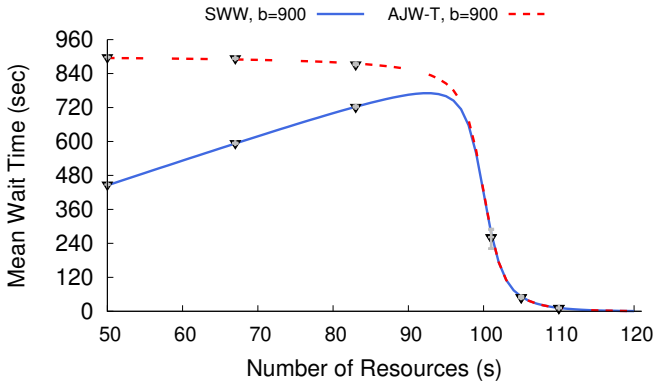


Fig. 6. Mean waiting time as a function of fixed resources under SWW and AJW-T where b=900s=15m.

3.3 All Jobs Wait - Threshold

Model Analysis. AJW and NJW define two extremes: AJW yields a low price but with a potentially high waiting time, while NJW yields a higher price but zero waiting time. The All Jobs Wait-Threshold (AJW-T) waiting policy defines a continuous tradeoff between these two extremes by requiring all jobs to wait up to some threshold time b, at which point the scheduler acquires an on-demand resource to service them. At b=0, AJW-T is equivalent to NJW, and as $b \to \infty$, AJW-T approaches AJW. To model AJW-T, we must derive r from Equation (4), or the fraction of jobs that run on on-demand resources after waiting b time. Given r, we can compute the effective price P from Equation (4) as before. In queuing literature, AJW-T is equivalent to a queuing model with reneging jobs that exit the queue after waiting a threshold period. The reneging probability r is given by the following lemma, which follows from an analysis by Liu and Kulkarni [27].

Lemma 3.1. The reneging probability r in a $M/M/s/\infty$ system is computed as follows.

$$r = \frac{\alpha \cdot \beta \cdot e^{-\delta \cdot b}}{s \cdot \mu} \tag{6}$$

where

$$\delta = (s\mu - \lambda) \tag{7}$$

$$\beta = \frac{s\mu p}{1-p} \tag{8}$$

$$p = \frac{(\lambda/\mu)^s}{s! \sum_{i=0}^{s} \frac{(\lambda/\mu)^i}{i!}} \tag{9}$$

$$\alpha = \begin{cases} [\beta(\frac{1}{\delta} - e^{\delta \cdot b} \cdot \frac{\lambda}{\delta \cdot s\mu}) + 1]^{-1} & \rho \neq 1 \\ \frac{\lambda}{\lambda + \beta \cdot (\lambda \cdot b + 1)} & \rho = 1 \end{cases} \tag{10}$$

When expanded, r is solely a function of s, b, λ, and μ. As before, we need an expression for the mean utilization of the s^{th} resource, as in Equation (3), to solve for the optimal s that minimizes cost. However, in this case, we replace Erlang's B formula with r above when using $s-1$ and s resources, as shown below, since r represents the reneging probability under AJW-T, which is akin to the blocking probability under AJW. We can again solve for the optimal s that minimizes price numerically using a binary search, as ρ_s is still monotonically decreasing as s increases, where a=λ/μ.

$$\rho_s = a \times [r_{s-1} - r_s] \tag{11}$$

After determining the optimal s and r for a given threshold waiting time b, we compute the mean waiting time of jobs. Liu and Kulkarni give the mean waiting time under reneging as follows [27]. The first additive term represents the mean waiting time for the jobs that execute on fixed resources, while the second additive term represents the mean waiting time for jobs that execute on on-demand resources, which is simply $r \times b$ as they all wait the maximum time b.

$$w = \begin{cases} (1-r) \times (\frac{\alpha \times \beta (1 - \delta b e^{-\delta \times b} - e^{-\delta \times b})}{(1-r) \times \delta^2}) + r \times b & \rho \neq 1 \\ (1-r) \times (\frac{\alpha \times \beta \times b^2}{(1-r) \times 2}) + r \times b & \rho = 1 \end{cases} \tag{12}$$

Empirical Validation. We again validate our model using our baseline parameters. Figure 4 shows the effective price P as a function of fixed resources s under AJW-T for different threshold maximum waiting times b, as well as the price under AJW and NJW. Once again, the model's predictions closely match the empirical results. As expected, as b increases, the price approaches AJW, and as it decrease the price approaches NJW. The graph also shows that as b increases, the optimal fixed resources s that minimizes price decreases. Similarly, Figure 5 shows the mean waiting time w on the y-axis as a function of the fixed resources s. Here, as b increases, the mean waiting time increases more sharply as $s \to 100$. Thus, unlike AJW and NJW, AJW-T is configurable, enabling users to set their tradeoff between price and waiting time.

Key Point. *AJW-T offers a configurable tradeoff between price and waiting time by enabling users to set the maximum waiting time threshold b, unlike NJW, which offers no tradeoff, and AJW, which offers a risky tradeoff.*

4 Selective Waiting Policies

Unlike non-selective waiting policies, selective waiting policies do not apply to all jobs, but only to selected jobs based on system or job characteristics. We define and analyze two selective policies: Short Waits Wait (SWW) and Long Jobs Wait (LJW). Since waiting policies are not mutually exclusive, we also analyze a compound waiting policy that combines SWW, LJW, and the threshold waiting time from AJW-T.

4.1 Short Waits Wait

Model Analysis. Unlike AJW-T where jobs wait up to a threshold value before they are scheduled on on-demand resources, in the Short Waits Wait (SWW) waiting policy, incoming jobs *estimate* their waiting time upon arrival (based on the jobs running and ahead of it in the queue) and only wait if the estimated wait time

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems

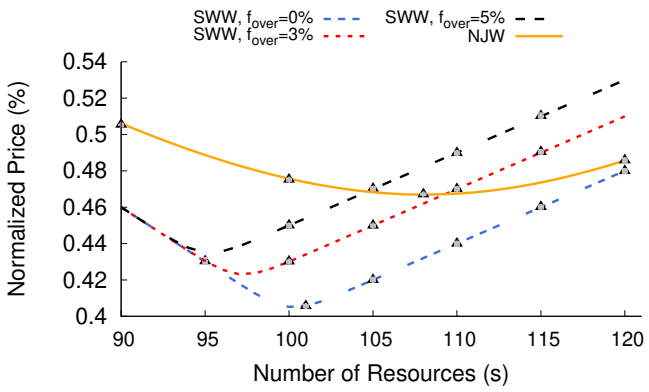


Fig. 7. Price P as a function of fixed resources s under SWW for different over-prediction errors f_{over} and NJW.

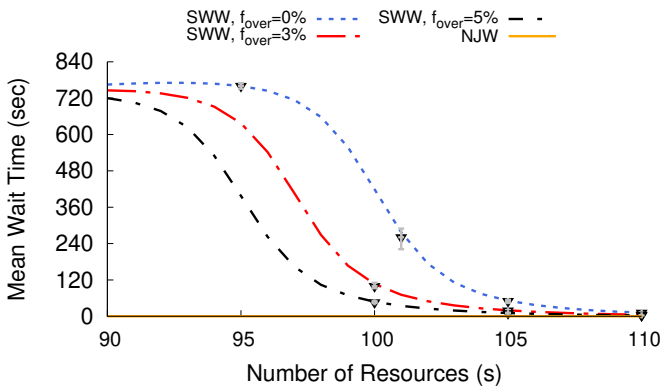


Fig. 8. Mean waiting time as a function of s under SWW for different over-prediction errors f_{over} and NJW.

is short, i.e., less than a threshold value. If the estimated wait time is long, i.e, exceeds the threshold, then they immediately run on on-demand resources without waiting. In queuing literature, this behavior is equivalent to a queuing system with balking jobs, which immediately exit the system if the waiting time will exceed a maximum threshold value denoted by b. Importantly, as prior work shows, the same set of jobs that renege under AJW-T, and in our case run on on-demand resources, will also balk under SWW [27]. Thus, the fraction of jobs r that run on on-demand resources under SWW is the same as under AJW-T (from Lemma 3.1), and thus the effective price for resources is the same under AJW-T and SWW for the same b.

The only change with SWW relative to AJW-T is the mean waiting time w, since under SWW jobs exit the system immediately and run on on-demand resources if their waiting time *would* exceed the threshold waiting time b. In this case, the mean waiting time w shown below is the same as in Equation (12) except that we remove the $r \times b$ term, since the r fraction of jobs that run on on-demand resources incur no waiting time rather than incurring b waiting time, as in AJW-T.

$$w = \begin{cases} (1-r) \times \left(\frac{\alpha \times \beta (1 - \delta b e^{-\delta \times b} - e^{-\delta \times b})}{(1-r) \times \delta^2}\right) & \rho \neq 1 \\ (1-r) \times \left(\frac{\alpha \times \beta \times b^2}{(1-r) \times 2}\right) & \rho = 1 \end{cases} \quad (13)$$

Empirical Validation. Figure 6 plots the mean waiting time w for SWW and AJW-T as a function of the fixed resources s, and a threshold waiting time b=900s=15m. The mean waiting time for SWW approaches zero as s decreases (and load increases) rather than b for AJW-T, as increasingly more jobs exit the system without waiting and run on on-demand resources. Note that SWW's mean waiting time reaches its maximum at s=93, and is always less than that of AJW-T.

Key Point. *SWW with accurate predictions of job waiting time is strictly better than AJW-T for the same threshold b, yielding the same price at a lower mean waiting time.*

4.1.1 Prediction Accuracy

The SWW analysis above assumes that arriving jobs are able to perfectly predict their waiting time w. Doing so requires perfectly predicting the running time of every job currently running and ahead of them in the queue. There is significant prior work on predicting queue waiting times using statistical analyses and machine learning classifiers, which we discuss in §8. This prior work demonstrates that accurately predicting queue waiting times can be challenging. As a result, we also model and analyze SWW under inaccurate predictions of job waiting time. Our analysis provides a basis for understanding how accurate machine learning (ML) classifiers and other methods that predict job waiting time developed in prior work must be to achieve specific job waiting time or cost targets. Importantly, the goal of our work is not to develop a better waiting time predictor, but to understand both how inaccurate predictions can affect waiting policies, and how to reason about the effectiveness of prediction methods.

Given a threshold waiting time b, there are two misprediction cases to consider: the scheduler either i) over-predicts a job's waiting time and thus runs it on on-demand resources when it should have waited for fixed resources, or ii) under-predicts a job's waiting time and thus forces it to wait for fixed resources when it should have run immediately on on-demand resources. We consider each case separately based on the fraction of jobs f_{over} and f_{under} that over- and under-predict their waiting time, respectively. Note that our analyses for over- and under-predicting waiting time can be applied independently to the same model. Of course, by definition, the set of jobs in f_{over} and f_{under} must be disjoint and $f_{over} + f_{under} \leq 1$.

Over-predicting Waiting Time. As the fraction of jobs that over-predict waiting time approaches 100%, SWW approaches the behavior of using all on-demand resources (plus the cost of the fixed resources), as jobs always immediately exit the system (due to their high predicted waiting time) and run on on-demand resources. For simplicity, our analysis here is not work-conserving, such that over-predictions redirect jobs to on-demand resources even when fixed resources are available. We can model over-predictions by simply reducing the arrival rate λ of jobs to the s fixed resources by f_{over}, since this fraction of jobs always exit the system due to over-prediction and run on on-demand resources. Thus, the effective arrival rate becomes $(1 - f_{over}) \times \lambda$. We can then solve for the optimal s as before using Equation (11) but substituting this new effective arrival rate for λ. We must also account for the increased f_{over} fraction of jobs that run on on-demand resources when computing the effective price P. To do so, we adjust Equation (4) as shown below.

$$P = (1-r) \times \frac{p_f}{\rho_f} + (1 - f_{over}) \times r \times p_o + f_{over} \times p_o \quad (14)$$

The first term is the same as in Equation (4). The second term represents the fraction of offered load that runs on on-demand resources after correct predictions, while the third term represents the fraction that runs on on-demand resources after incorrect over-predictions. We similarly adjust the waiting time in Equation (13)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems

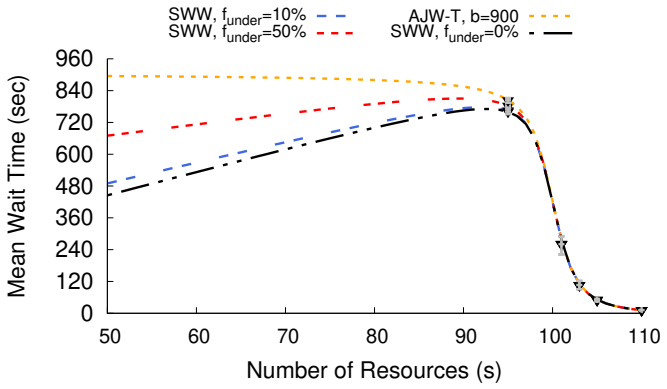


Fig. 9. Mean waiting time as a function of fixed resources s under SWW for different under prediction rates f_{under}.

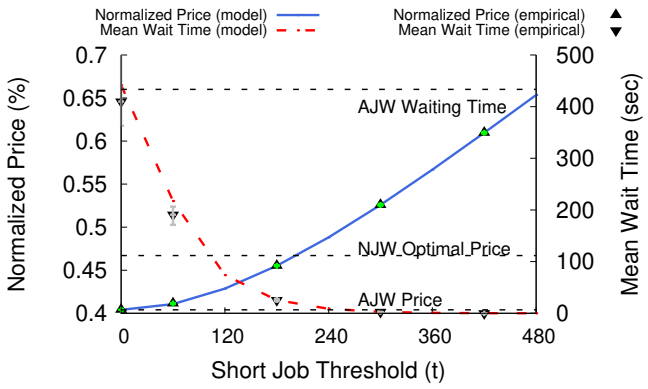


Fig. 10. Normalized price P and mean wait time w as a function of the short job threshold t (in seconds) for s=101 under an LJW waiting policy.

by substituting $(1-r)$ with $(1-f_{over})\times(1-r)$ as fewer jobs wait for fixed resources.

Figure 7 shows the effective price P (normalized by the on-demand price as before) on the y-axis as a function of s. In this case, we use the same baseline parameters as before, while setting b=900, and plot different lines for different values of f_{over}, as well as NJW. As the graph shows, as f_{over} increases to one, the optimal value of s changes, and approaches that under NJW. Note that the price of a work-conserving variant would be bounded by NJW as s increases, rather than exceeding it, since it would utilize any idle fixed resources. However, the behavior would be the same as in the graph as s decreases, since there are fewer idle fixed resources. Similarly, Figure 8 shows the mean waiting time w as a function of s. As expected, as f_{over} increases, the mean waiting time decreases (as fewer jobs wait). As before, we include both continuous functions from our model and empirical results from our job simulator.

Key Point. *SWW is sensitive to over-predictions, as 3-5% over-predictions significantly alters the price and mean waiting time.*

Under-predicting Waiting Time. As the fraction of jobs f_{under} that under-predict their waiting time approaches 100%, SWW approaches the behavior of AJW-T, since jobs always wait for fixed resources up to threshold b before running on on-demand resources. We model this case by using the fact that the set of reneging jobs under AJW-T and balking jobs under SWW are the same [27], and thus do not affect the waiting time of other jobs. The f_{under} fraction of jobs that should balk and run immediately on on-demand resources due to a long wait time, but instead wait due to an under-prediction, will always eventually renege and run on on-demand resources. Since these jobs never run on fixed resources, they do not affect the waiting times of the jobs that do.

The effective price under SWW with under-predictions is the same as that with AJW-T and SWW with perfect predictions, as the same set of jobs run on on-demand resources in all cases. The only difference is the job waiting times. To compute the waiting time in this case, we simply need to substitute $(1-f_{under})\times r$ for r in Equation (12) for the waiting time under AJW-T to account for the fraction of jobs f_{under} that incur a waiting time of b due to the under-prediction. Figure 9 shows the mean waiting time w as a function of s using our baseline parameters for different values of f_{under}, as well as for AJW-T with b=900s. As expected, as f_{under} increases, the mean waiting time increases until it matches that of AJW-T. Notably, SWW is much less sensitive to under-predictions, since they do not affect price and only affect the mean waiting time when fixed resources are highly under-provisioned. Further, even when under-provisioned, the under-prediction rate must be high, at >50% in the graph, to cause a significant increase in mean wait time.

Key Point. *SWW is not highly sensitive to under-predictions, as they do not affect the effective price and only affect the mean waiting time when fixed resources are under-provisioned.*

Our results are important in assessing and contextualizing the accuracy of new and existing methods for predicting queue waiting times. Specifically, for cloud-enabled schedulers, these prediction techniques should focus on minimizing over-predictions, and they should be evaluated separately for over- and under-predictions.

4.2 Long Jobs Wait

Model Analysis. Long Jobs Wait's (LJW) intuition is that longer running jobs should be willing to wait longer for fixed resources, since longer waiting times are a smaller percentage of their overall running time compared to shorter jobs. For LJW, we introduce a running time threshold t such that jobs shorter than t run immediately on on-demand resources, while others wait for fixed resources. For simplicity, our LJW policy is not work-conserving in that it runs short jobs on on-demand resources even if fixed resources are available. This non-work-conserving variant will behave similarly to a work-conserving one in the optimal case when fixed resources are not over-provisioned (and thus rarely idle). For LJW, we separate the analysis for short jobs and long jobs. As shown below, the effective price P is the weighted average of the price to run short and long jobs. As before, r represents the fraction of jobs that run on on-demand resources, while P_{short} and P_{long} represent the price to run short and long jobs, and μ_{short} and μ_{long} represent the mean service rate of short and long jobs.

$$P = (1-r) \times \frac{\mu}{\mu_{long}} \times P_{long} + r \times \frac{\mu}{\mu_{short}} \times P_{short} \quad (15)$$

Thus, first and second additive terms represent the relative cost to execute long and short jobs, respectively, based on their fraction of the total jobs, their proportion of the service time, and their price. Note that, $\mu_{long} > \mu > \mu_{short}$ for any $t > 0$. Similarly, the mean waiting time w is the weighted average of the waiting time to run short and long jobs. Since, by definition, short jobs do not wait, w is only dependent on the fraction of long jobs and their mean waiting time.

$$w = (1-r) \times w_{long} \quad (16)$$

Short Jobs. All short jobs (with running times $<t$) run on on-demand resources at price p_o without any waiting time. Thus,

P_{short}=p_o, while r is the fraction of jobs with running times less than t, which is equivalent to the CDF of the exponential distribution for service times at x=t, as shown below.

$$r = 1 - e^{-\mu t} \tag{17}$$

Long Jobs. Since long jobs always wait for fixed resources, the policy is similar to AJW in §3.1 but applied to long jobs. The mean arrival rate for long jobs λ_{long} is the product of the overall job arrival rate λ and the fraction of long jobs $(1-r)$.

$$\lambda_{long} = \lambda \times (1-r) = \lambda \times e^{-\mu t} \tag{18}$$

Similarly, we compute the mean service rate μ_{long} for long jobs using its service time PDF $f(x,\mu)$, as below. The PDF for long jobs is an exponential distribution shifted by t units.

$$f(x,\mu) = \mu e^{-\mu(x-t)}, x \geq t \tag{19}$$

We find the expected value of the long jobs service time PDF to derive its mean service time $\frac{1}{\mu_{long}}$ by integrating from x=t→∞.

$$\frac{1}{\mu_{long}} = \int_t^{\infty} x\mu e^{-\mu(x-t)} dx = t + \frac{1}{\mu} \tag{20}$$

Note that we can derive μ_{short} from μ_{long}, r, and μ, since the mean service time of the original distribution $1/\mu$ is the weighted average of the mean service time of short jobs $1/\mu_{short}$ and long jobs $1/\mu_{long}$. Thus, we compute μ_{short} by simply solving the expression below.

$$\frac{1}{\mu} = r \times \frac{1}{\mu_{short}} + (1-r) \times \frac{1}{\mu_{long}} \tag{21}$$

The effective price P_{long} of running long jobs on fixed resources is simply the price of fixed resources p_f at full utilization divided by the actual utilization ρ_{long}, where $\rho_{long} = \lambda_{long}/(s \times \mu_{long})$.

$$P_{long} = \frac{p_f}{\rho_f} = \frac{p_f \times s \times \mu_{long}}{\lambda_{long}} \tag{22}$$

Importantly, however, the distribution of jobs with service times greater than t is *not* exponentially distributed. As a result, we *cannot* apply the same model as for AJW to compute the waiting time. Instead, we use the well-known approximation below for the waiting time of an M/G/s queue, where CV is the distribution's coefficient of variation, i.e., the standard deviation divided by the mean. In this case, the standard deviation of the long jobs' service time distribution is $1/\mu$, and the mean is $1/\mu_{long}$, so CV=μ_{long}/μ.

$$w \sim \frac{CV^2+1}{2} \times \frac{C(s,a)}{s \times \mu_{long} - \lambda_{long}} \tag{23}$$

Empirical Validation. Figure 10 shows the normalized price (left y-axis) and waiting time (right y-axis) under LJW as a function of t for s=101, as well as AJW and NJW, using our baseline parameters. As before, the graph shows that the empirical values closely match the model's waiting time approximation above. The graph shows that as t increases the normalized price increases, as fewer jobs wait for resources. However, LJW also significantly decreases the mean waiting time relative to AJW as t increases, since the exponential service time distribution is weighted towards short jobs, which experience no waiting time under LJW. In addition, since long jobs still comprise a high fraction of the overall service time (and thus cost), the effective price under LJW, especially for small values of t, increases at a much lower rate than the waiting time decreases. For example, at a threshold t=180, the mean wait time is near 0 under LJW compared to a mean waiting time of 450s under AJW, for a normalized price that is only ∼10% higher, but slightly lower than NJW.

By immediately running short jobs, LJW acts as the dual of shortest job first scheduling that minimizes waiting time, and is thus beneficial when fixed resources are under-provisioned.

Key Point. *LJW offers a nice tradeoff: as t increases, price increases modestly, while waiting time decreases significantly.*

4.2.1 Prediction Accuracy

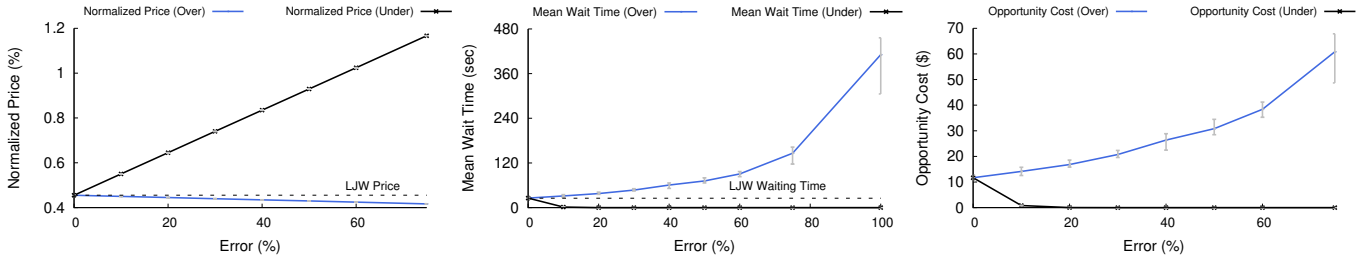
Our LJW analysis above assumes that arriving jobs perfectly predict their running time, which may not always be possible. As with predictions of queue waiting time, there is significant prior work on predicting job running time, which we discuss in §8, since it is an important input for many common scheduling policies, such as SJF. As in 4.1.1, our analysis provides a basis for contextualizing this prior work, and understanding how inaccuracy can affect waiting policies. Since an imperfect prediction analysis for LJW is more challenging than for SWW, we empirically quantify the effect of inaccurate predictions of job running time in our model. At a high level, similar to SWW's analysis, as f_{over}—the fraction of short jobs that are predicted to be long (with running times $>t$)—approaches one, LJW approaches the behavior of AJW, since all jobs wait. In contrast, as f_{under}—the fraction of long jobs that are predicted to be short—approaches one, LJW approaches using all on-demand resources (plus the cost of fixed resources).

To understand how sensitive LJW is to over- and under-predictions of job running time, we plot the normalized price and mean waiting time as a function of f_{under} and f_{over} for s=101 and t=180. We only plot empirical results from our job simulator, since we have no analytical model. Figure 11(a) shows that as the over-prediction rate increases, the effective price decreases, but, since LJW's price in this case is already near the optimal price p_f, the decrease is minimal. In contrast, as the under-prediction rate increases, the effective price increases significantly. Figure 11(b) shows the opposite effect: as the over-prediction rate increases, the mean waiting time increases significantly, while as the under-prediction rate increases the mean waiting time decreases, although since LJW's mean wait time is already near zero, the decrease is not significant.

To clarify the tradeoff between over- and under-prediction, we define a new metric, called the *opportunity cost of waiting*, which values a job's waiting time equal to its running time. The mean opportunity cost $P \times w$ is in dollars. Since lower values of P and w are better, a lower opportunity cost is better. Figure 11(c) shows the opportunity cost of LJW as a function of the rate of over- and under-prediction. The graph shows that LJW is more robust to under-prediction, since a high under-prediction error rate causes more jobs to run on on-demand resources. While this increases the price, it drops the waiting time (and thus opportunity cost) to zero once the error rate exceeds 10%. In contrast, over-predictions decrease the price only linearly, as shown in (a), but increases the waiting time super-linearly, as shown in (b). Over-predictions cause more jobs to wait on fixed resources, which significantly increases the queue length and waiting time, for only a modest cost savings. The result is a super-linear increase in opportunity cost, as the super-linear increase in waiting time dominates the linear decrease in price.

Key Point. *LJW's effective price is robust to over-predictions and sensitive to under-predictions, while its mean waiting time is robust to under-predictions and sensitive to over-predictions. LJW*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems



(a) Normalized Price (b) Mean Job Waiting Time (c) Opportunity Cost

Fig. 11. Normalized price (a), mean job waiting time (b), and opportunity cost (c) as a function of the fraction of jobs with incorrect over- and under-predictions (%) of job running time for s=101 and t=180 under an LJW waiting policy.

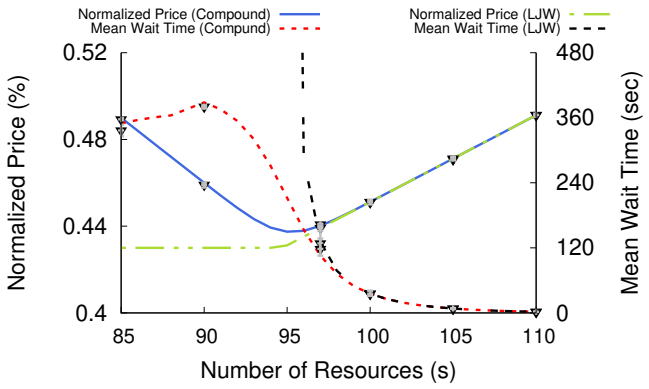


Fig. 12. Normalized price P and wait time w as a function of fixed resources s for our compound policy (b=900 and t=180) and LJW (t=180).

is more sensitive to over-predictions, since they cause a super-linear increase in waiting time for only a linear decrease in price.

Similar to SWW's uncertainty analysis in §4.1.1, our results above are important in assessing and contextualizing the accuracy of new and existing methods for predicting job running times. Specifically, for cloud-enabled schedulers, these prediction techniques should focus on minimizing over-predictions, and they should be evaluated separately for over- and under-predictions.

4.3 Compound Waiting Policies

Model Analysis. Waiting policies, unlike scheduling policies, are not mutually exclusive. That is, we can concurrently apply multiple waiting policies that select jobs to wait based on different characteristics. Thus, we analyze a compound waiting policy that combines the advantages of AJW-T, SWW, and LJW. In analyzing this policy, we first apply LJW's analysis from §4.2, since its waiting decisions are based on job running time, and are thus load insensitive and not affected by other waiting policies. Our LJW analysis yields a fraction r of short jobs that always run on on-demand resources, which we label r_{short}. The remaining $(1-r_{short})$ long jobs run on fixed or on-demand resources depending on their waiting time.

We next apply SWW's analysis from §4.1 solely to the remaining long jobs. In particular, we compute the fraction r_{sww} of the remaining long jobs that run on on-demand resources (due to long wait times) by applying Lemma 3.1 using λ_{long} and μ_{long} from §4.2 for a given value of s and b. This is an approximation, since Lemma 3.1 assumes exponentially distributed service times, and the long jobs' service time distribution is an exponential distribution truncated at t. This approximation becomes more accurate as $t\to 0$ and the distribution approaches an exponential. Given r_{sww}, the effective price for our compound waiting policy is as follows.

$$P = (1-r_{short}) \times (1-r_{sww}) \times \frac{\mu}{\mu_{long}} \times \frac{p_f}{\rho_f} + (1-r_{short}) \times r_{sww} \times \frac{\mu}{\mu_{long}} \times p_o + r_{short} \times \frac{\mu}{\mu_{short}} \times p_o \tag{24}$$

The last additive term is the product of the fraction of short jobs that run on on-demand resources, their fraction of the mean service time, and the on-demand price. The second term is the same, but applies only to the fraction of long jobs with high wait times that run on on-demand resources. The first additive term is the remaining long jobs with short waiting times that run on fixed resources. Here, ρ_f, shown below, is the mean utilization of the fixed resources, which is simply the adjusted arrival rate of jobs to the fixed resources divided by their mean service rate, and then normalized by s.

$$\rho_f = \frac{(1-r_{short}) \times (1-r_{sww}) \times \lambda}{s \times \mu_{long}} \tag{25}$$

We use the same approach as in LJW to approximate the compound policy's mean waiting time, but replace the waiting time under AJW with the waiting time under SWW from Equation (13) as below, again using λ_{long} and μ_{long} as the input. The coefficient of variation CV is the same as in LJW.

$$w \sim \begin{cases} \frac{CV^2+1}{2} \times (1-r_{sww}) \times \left(\frac{\alpha \times \beta (1-\delta b e^{-\delta \times b} - e^{-\delta \times b})}{(1-r_{sww}) \times \delta^2}\right) & \rho < 1 \\ \frac{CV^2+1}{2} \times (1-r_{sww}) \times \left(\frac{\alpha \times \beta \times b^2}{(1-r_{sww}) \times 2}\right) & \rho = 1 \end{cases} \tag{26}$$

Empirical Validation. Figure 12 compares our compound waiting policy with LJW using our baseline parameters with b=900 and t=180. The primary advantage of the compound policy over LJW is that it strictly lowers the overall waiting time, since long jobs do not wait indefinitely, which is especially important when resources are under-provisioned, for nearly the same effective price. As shown, the compound policy's mean waiting is less than or equal to that of the LJW policy.

Key Point. *Our compound policy combines the advantages of AJW-T, SWW, and LJW, and thus offers the best tradeoff.*

4.4 Model Results Summary

Our analyses show that waiting policies offer a complex tradeoff between fixed resource provisioning, cost, and waiting time. To summarize these tradeoffs, we again use the opportunity cost of waiting. Recall from §4.1.1 that the mean opportunity cost equals $P \times w$ and is in dollars, where lower values of P and w are better. Figure 13 shows the mean opportunity cost of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems

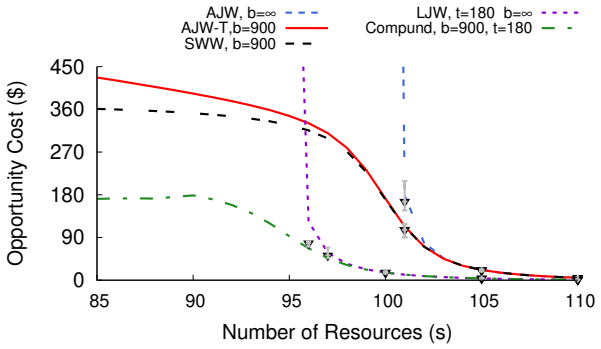


Fig. 13. Opportunity cost as a function of fixed resources s under AJW, AJW-T, SWW, LJW, and compound policy when using *FCFS scheduling*.

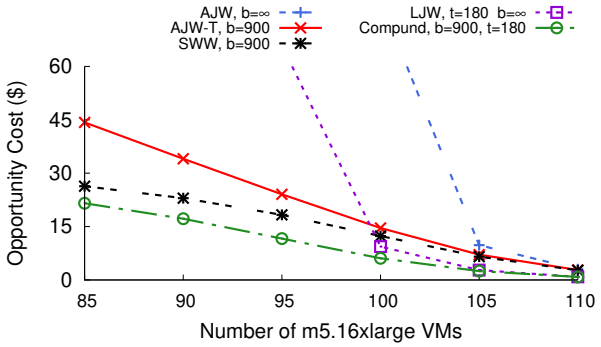


Fig. 14. Opportunity cost as a function of fixed resources s under AJW, AJW-T, SWW, LJW, and compound policy when using *SJF scheduling*.

waiting for AJW, AJW-T (for b=900), SWW (for b=900), LJW (for t=180), and our compound policy (for b=900 and t=180) using our baseline parameters. Since the effective price P is bounded (by p_f) and waiting time is not, the opportunity cost for all policies approaches zero as s increases. Just as with a scheduling policy, a waiting policy's importance increases with resource constraint. We exclude NJW, as its opportunity cost is always zero, since its waiting time is zero. For the remaining policies where a price-waiting time tradeoff exists, our compound policy yields the lowest opportunity cost.

While the inter-arrival and service time distributions affect the absolute differences in price and waiting time between waiting policies, many aspects of our analysis are generalizable, and hold regardless of the job inter-arrival and service time distributions. As a result, our models and analysis are most useful in enabling users to better understand and reason about the effect of different waiting policies by understanding how their system and workload characteristics differ from the models' assumptions. Specifically, SWW always results in a shorter mean waiting time than AJW-T; higher values of the waiting time threshold b always increase fixed resource utilization, decrease price, and increase waiting time; increasing the short job threshold t always increases price and decreases waiting time; and the compound policy always combines the advantages of AJW-T, SWW, and LJW. Our evaluation in §6 echoes this point by empirically showing the relative price, waiting time, and opportunity cost between the waiting policies of a real workload precisely follows our analysis.

In addition, the general insights above also hold for different scheduling policies. While the waiting policy is distinct from the scheduling policy, and both can be defined independently, there is some interaction between them. Figure 14 shows the same experiment as Figure 13, but with shortest job first (SJF) as the scheduling policy instead of FCFS. Note that Figure 14 only plots data from simulations of the same synthetic workloads as in Figure 13, since there is no similar closed-form analytical queuing models for SJF scheduling. In addition, predictions of queue waiting time under SWW are much more difficult under SJF, since the ordering of jobs in the queue changes based on newly arriving jobs. Thus, waiting time predictions under SJF require future knowledge. The graph shows that the relative ordering of waiting policies is the same when using SJF and FCFS, and also that the trends are the same. Of course, the absolute opportunity cost when using SJF is significantly less because SJF substantially decreases the waiting time for jobs that wait for fixed resources.

As we show in §6.2, the price when using SJF is similar to using FCFS. Using SJF has no effect on LJW, since it determines whether a job waits based on its own characteristics. SJF does affect SWW: since short jobs wait less than long jobs under SJF, SWW in this case prioritizes short jobs to wait for fixed resources. However, in isolation, prioritizing short jobs does not substantially increase the price, since most jobs are short anyway (both in the synthetic workloads here and our real workload in §6), a similar set of jobs run on on-demand VMs. When SWW is used in conjunction with LJW under SJF scheduling, LJW cancels out this implicit short job prioritization of SWW under SJF, because LJW automatically sends short jobs to run on on-demand resources. We discuss the inter-play between the waiting policy and scheduling policy in the context of SJF more in §6.2.

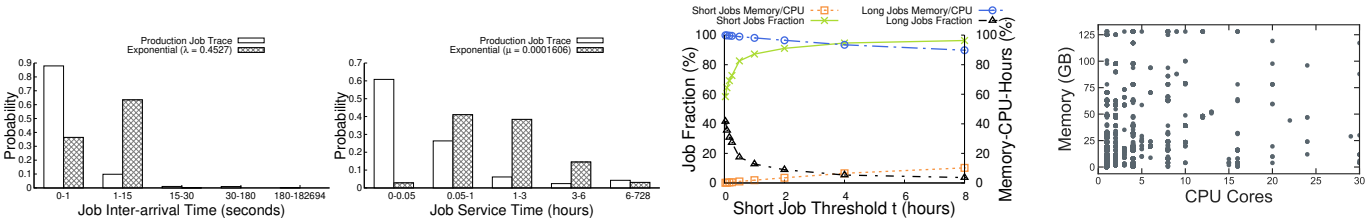
5 Implementation

We implemented a waiting policy model analyzer based on our analysis, as well as a trace-driven job simulator, in python.

Model Analyzer. Our model analyzer implements the analytical queuing model for all the waiting policies we analyze. The analyzer enables what-if analyses to compare and understand a workload's expected cost and job waiting times under different policies and parameter values. The analyzer takes as input a policy's name and λ, μ, s, p_f, and p_o, as well as b for AJW-T, SWW, and the compound policy, and t for LJW and the compound policy. Users may also enter a workload duration T. The analyzer's output is the policy's mean waiting time w, the effective price P, the fraction of jobs that run on on-demand resources r, and, if T is specified, the total cost C. If s is unspecified, the analyzer finds the optimal s that minimizes price P and outputs the values above at the optimal. We plan to publicly release our model analyzer, which can be used to re-produce our model graphs in §3 and §4.

Job Simulator. We implemented a trace-driven job simulator in python that mimics a cloud-enabled job scheduler, which can acquire VMs on-demand to service jobs. The simulator uses a FCFS scheduling policy, and also implements each of our waiting policies. The simulator takes as input a trace of jobs, s, p_f, the name of a waiting policy, and the same waiting policy-specific parameters as above. Users must also specify the number of cores and memory allotment for each fixed resource s. Since cloud platforms offer VMs in different sizes, the simulator includes a table of available on-demand VM options that specify their cores, memory, and price. In our evaluation, we consider only the 8 VM types in the `m5` family of general-purpose VMs on AWS. While VMs in the `m5` family have different resources, they all offer the same price per unit of resource. The simulator's output is the mean waiting time w, the effective price P, the fraction of jobs that run on on-demand resources r, and the total cost C.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems



(a) Job Interarrival Times (b) Job Service Times (c) Job Resource Requirements (d) Long and Short Job Mix

Fig. 15. Histograms of job inter-arrival times (a) and service times (b) for our real production batch workload along with an exponential distribution using the same mean, as well as the mix of long and short jobs (c) and a scatterplot of job resource requirements (d).

Each job in the input trace has a service time based on its resource, e.g., core and memory, requirements. For fixed resources, we use only `m5.16xlarge` VMs—the largest in the `m5` family—to mitigate the impact of imperfect packing of jobs to VMs. Our job simulator packs jobs onto fixed resources using a simple best-fit heuristic. When selecting an on-demand VM for a job, our simulator selects the smallest (and cheapest) VM from among the 8 types within the `m5` family to run the job that satisfies the job's resource requirements. Note that our analytical model of LJW in §4.2 is not work-conserving. For consistency, our LJW results in §6 are also not work-conserving, although our simulator can also be configured to be work-conserving. Our job simulator also does not include some system characteristics that may be important in practice, such as startup overheads and additional external constraints on which VMs each job may use, since our goal is not to precisely replicate the behavior of a real system, but to isolate and understand the impact of different waiting policies on price and waiting time. The effect of these additional system characteristics are orthogonal to the waiting policy, and including them would serve to obscure, rather than reveal, the effects of the waiting policy. We have publicly released our job simulator at the UMass Trace Repository [7], [9].

Real-world Data. In §6, we use our job simulator to quantify the impact of different waiting policies on a real year-long job trace that includes 14M jobs from a production high-performance computing cluster consisting of 14.3k cores. The cluster is the University of Massachusetts (UMass) System Shared Cluster, and is available for general use to researchers from all five campuses in the UMass system [8]. Thus, the workload is a representative sample of job types across the entire scientific, engineering, and medical research communities. The cluster is housed in the Massachusetts Green High Performance Computing Center (MGHPCC), a 15MW data center in Holyoke, Massachusetts that also hosts infrastructure Boston University, Harvard, MIT, and Northeastern. The cluster runs the LSF job scheduler, and we use its log from the year 2016 to drive our simulations. Each job entry in the log includes its submission time, user ID, maximum running time limit, requested number of cores and memory, and running time. Note that the maximum running time limit is not an effective predictor of job running time, since it is typically many orders of magnitude greater than jobs' actual running time. As a result, using the maximum running time as a predictor results in nearly 100% of jobs having their running time under-predicted. We modify the raw trace to conform to our job simulator's input format. We have publicly released this job trace [7], [10].

6 Evaluation

We do not intend our models to be predictive, but instead evaluate their usefulness in analyzing a real year-long batch workload. Specifically, we show that our models both 1) accurately predict the relative price and waiting time between different waiting policies in our real workload, and 2) enable reasoning about price and waiting time by understanding the differences between our model's and the real workload.

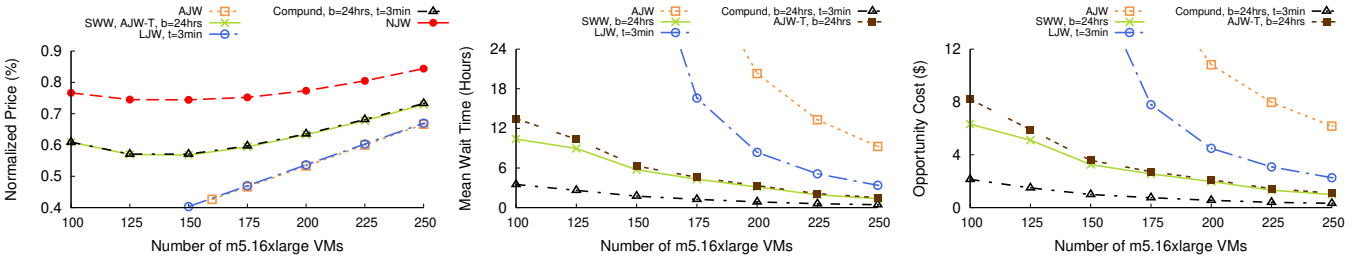
6.1 Workload Characteristics

Figure 15 characterizes our real workload and our model's ideal. Figure 15(a) is a histogram of job inter-arrival times for our trace and an exponential distribution with the same mean, which is 0.4527 jobs/sec. Note that the bin size is non-uniform, since our trace much more bursty than our model assumes. In particular, nearly 90% of job inter-arrival times are between 0 and 1 second compared with less than 40% for an exponential distribution with the same mean. An exponential distribution instead has more inter-arrival times between 1 and 15 seconds. Both distributions have a heavy tail with our job trace experiencing a few more extremely long inter-arrival times, between 3 minutes and 50 hours.

Figure 15(b) is a similar histogram of job service times with a mean service time $1/\mu$ of 6225 seconds (or 1.73 hours) per job. Again, the bin size is non-uniform due to our trace's large skew. In this case, over 60% of jobs are between 0 and 3 minutes, while an exponential distribution with the same mean has only 3% of its jobs in this range. Instead, the exponential distribution has more jobs of mid-range length between 3 minutes and 6 hours. However, our trace has a slightly higher fraction of extremely long jobs, which account for a large fraction of the overall job execution time and cost. Thus, overall, the job service times in our trace have both a heavier head and tail compared to the exponential distribution. To further illustrate, Figure 15(c) shows the fraction of long and short jobs, and their resource usage (in memory$\times$core hours), as a function of the short job threshold t. The graph shows that short jobs are a high fraction of jobs, even for large short job thresholds, but account for only a small fraction of the resource usage. As we show, since this skew is more extreme in our trace than in our model, LJW's ability to decrease mean waiting time is much greater than our model, since there is a larger fraction of short jobs that never wait.

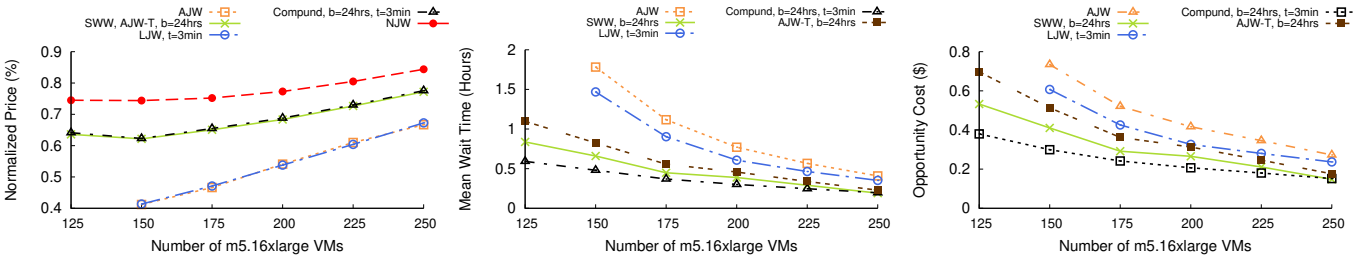
Finally, Figure 15(d) shows a scatterplot of the core and memory requirements for each job. Our model assumes job resource requirements are uniform and map directly to each VM's resources. However, our simulator only schedules a job on a VM if it has enough available cores and memory to satisfy a job's requirements. Our simulations assume a large `m5.16xlarge` VM with 64 cores and 256GB memory to mitigate imperfect job packing on VMs. We contextualize our results by comparing against the current fixed-size cluster, which consists of 14,376 cores and is equivalent to 225 `m5.16xlarge` VMs. Simulating this cluster on our trace yields a mean waiting time of 13.3 hours

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems



(a) Normalized Price (b) Mean Job Waiting Time (c) Opportunity Cost

Fig. 16. Normalized price (a), mean job waiting time (b), and opportunity cost (c) as a function of `m5.16xlarge` VMs when executing our real production batch workload under AJW, AJW-T, SWW, LJW, and our compound policy with *FCFS scheduling policy*.



(a) Normalized Price (b) Mean Job Waiting Time (c) Opportunity Cost

Fig. 17. Normalized price (a), mean job waiting time (b), and opportunity cost (c) as a function of `m5.16xlarge` VMs when executing our real production batch workload under AJW, AJW-T, SWW, LJW, and our compound policy with *SJF scheduling policy*.

and a cost of $2,421,965, or $276.48/hour. As before, we use a discount factor d∼0.4 based on the `m5.16xlarge`'s on-demand price of $3.072/hour and its 3-year reserved price of $16,046.

6.2 Real-world Workload Results

Figure 16 shows the normalized price (a), mean waiting time (b), and opportunity cost (c) for each of our waiting policies with FCFS scheduling policy. We select the maximum waiting time threshold b=24 hours for SWW and AJW-T, or slightly less than double the current cluster's mean waiting time using AJW. We select the long job cutoff t=3m where 60% of jobs are short and 40% are long.

Price. As expected, Figure 16(a) shows that AJW yields the lowest price, since it requires all jobs to wait for fixed resources. Interestingly, LJW yields nearly the same price even though it executes 60% of the total jobs on on-demand VMs. Since these 60% of short jobs comprise only a small fraction of the overall job execution time, executing them on on-demand VMs does not substantially increase the normalized price. SWW, AJW-T, and our compound policy yield nearly the same price for the same reason. This price is greater than LJW because SWW and the compound policy cut the tail off the job waiting time distribution by preventing jobs that would have to wait longer than 24 hours from ever waiting. Running these jobs, which may include long jobs, on on-demand VMs increases the price. As fixed resources decrease, the price reaches a minimum before increasing, as an increasingly larger share of the jobs experience (or would experience) long waiting times and thus instead run on on-demand resources. NJW has a ∼26% higher price than SWW, since it directs *any* job that must wait to on-demand resources.

When using AJW, our current cluster yields a normalized price of 0.6 at x=225 fixed resources, while the minimum cost under the compound policy is 0.571 at x=150, or 5% less. For our trace, P=0.6 translates to an annual cost of $2,421,965, while 0.571 translates to $2,304,903, or over $100k lower. This cost advantage for our compound policy is less than our model predicts, since our burstier workload causes more jobs to run on on-demand resources, which increases the price.

Waiting Time. As our model predicts, Figure 16(b) shows that the mean job waiting time under AJW and LJW increases super-linearly as fixed resources decrease. However, even though LJW's cost is nearly the same as AJW's, its mean waiting time is substantially less because the large fraction of short jobs never wait. In contrast, the mean waiting time under AJW-T, SWW, and the compound policy increases modestly as fixed resources decrease. Even at x=100, the mean waiting time of these policies is less than the 13.3 hour mean waiting time in our current fixed size cluster (AJW at x=225). At x=150, the compound policy has a mean waiting time of 1.74 hours, or 7× less than our current cluster (for 5% less cost).

Our compound policy's waiting time is much less than our model predicts due to the burstier workload, where large bursts of jobs cause long waiting times for a large fraction of short jobs under AJW. Running these short jobs on on-demand VMs significantly reduces waiting time at little cost. In addition, running jobs with long waiting times on on-demand VMs only modestly increases cost for large decreases in waiting time.

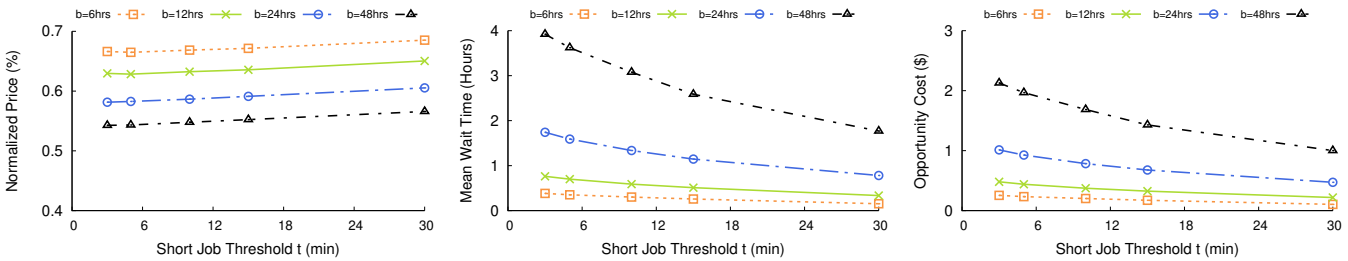
Opportunity Cost. Figure 16(c) graphs the mean opportunity cost of waiting $P \times w$ for each policy, and shows that, as our model predicts, the compound policy offers the best tradeoff by a significant margin compared to the other policies. Note that, even though our workload's characteristics differ significantly from those assumed by our model, the overall trends in opportunity cost match those from our model in Figure 13.

Key Point. *Our real workload's burstier job arrivals and heavier head/tail service time distribution makes the compound policy's waiting time advantage much greater than our model predicts.*

Key Result. *At the optimal, the compound policy decreases the cost (by 5%) and mean job waiting time (by 7×) compared to the current cluster using AJW, and decreases the cost (by 43%) compared to renting on-demand resources for a comparatively modest increase in mean job waiting time (at 1.74 hours).*

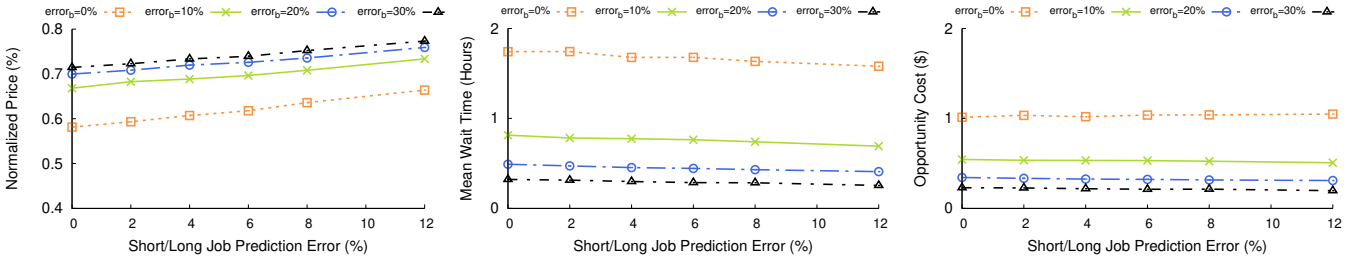
SJF Scheduling. We next repeat the experiments above using the same parameters but using the SJF scheduling policy instead of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems



(a) Normalized Price (b) Mean Job Waiting Time (c) Opportunity Cost

Fig. 18. Normalized price (a), mean job waiting time (b), and opportunity cost (c) as a function of the long job threshold (t) when executing our real production batch workload under a compound policy assuming 150 `m5.16xlarge VMs`.



(a) Normalized Price (b) Mean Job Waiting Time (c) Opportunity Cost

Fig. 19. Normalized price (a), mean job waiting time (b), and opportunity cost (c) as a function of the long job prediction error when executing our real production batch workload under a compound policy assuming 150 `m5.16xlarge VMs`.

FCFS scheduling. Figure 17 shows the results. As mentioned in §4.4, Figure 17(a) shows nearly the same normalized price across all the waiting policies as in Figure 16(a). In some cases, as with AJW and LJW, the price is the same, since these waiting policies are not sensitive to the scheduling policy. SWW is sensitive to the scheduling policy, and prioritizes short jobs on fixed resources, since these jobs have a lower waiting with SJF. However, since the vast majority of jobs in our real-world trace are already short, this only slightly increases the normalized price. Since the compound policy includes SWW, there is a similar impact on the normalized price. NJW's price is also higher under SJF for the same reason.

Figure 17(b) shows that SJF significantly decreases the waiting time across all waiting policies compared to Figure 16(b). Note that LJW and AJW in this experiment require a minimum of 150 VMs to run all jobs within the year, and thus we do not extend their results below 150 VMs. SJF is well-known to optimize for waiting time, often at the expense of starving longer jobs. However, in our case, long jobs never starve when using AJW-T, SWW, or the Compound policy, since in this case, if jobs have to wait longer than the waiting time threshold, the scheduler runs them immediately on on-demand. Importantly, the trends and relative ordering of the waiting policies under SJF is the same as under FCFS based on our analysis from §4.4.

Finally, Figure 17(c) shows the opportunity cost of all waiting policies under SJF. As with our model's workload in §4.4, the opportunity cost decreases compared to FCFS due to the substantial decrease in waiting time. As when using FCFS, the relative ordering of opportunity cost when using SFJ remains the same with the compound policy yielding the lowest opportunity cost.

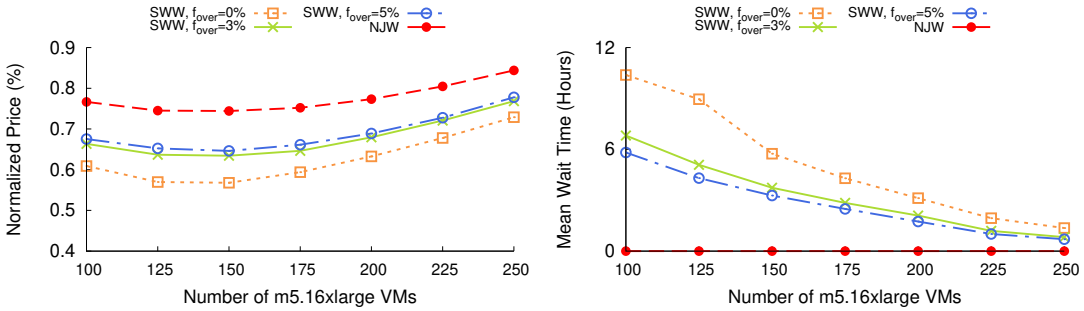
6.3 Sensitivity Analysis

We perform a sensitivity analysis that varies b, t, and errors in estimating job waiting time and running time to understand their effect on the results. We chose the values above for b=24h and t=3m arbitrarily to be reasonable, as 24h is roughly twice the mean waiting time under AJW and t=3m categorizes a large fraction (60%) of jobs as short. We also assume accurate estimates of job waiting and running time, e.g., using historical data. Our sensitivity analysis assumes 150 `m5.16xlarge`'s when using the compound policy, which as discussed in §6.2 and shown in Figure 16(a) and Figure 17(a) is the number of fixed resources that minimizes cost under both FCFS and SJF scheduling, respectively.

Parameter Sensitivity. Figure 18 plots price, waiting time, and opportunity cost as a function of the short job threshold t with lines for different values of the waiting time threshold b. We vary t from 3-30m and the waiting time threshold from 6h-48h. The price (a) increases linearly with the short job threshold t, albeit with a small slope, since this increases the fraction of short jobs that run on on-demand VMs at a higher price. The price also decreases roughly linearly for every doubling of the waiting time threshold b, as longer waiting time thresholds force more jobs to wait for lower cost fixed VMs. In contrast, the mean waiting time (b) decreases as the short job threshold increases, at an increasingly slower rate, as fewer jobs wait for fixed VMs. This non-linearity derives from Figure 15(c). Similarly, the mean waiting time decreases as the waiting time threshold decreases, also at an increasingly slower rate. Finally, the opportunity cost (c) is dominated by the mean waiting time, and thus exhibits a similar trend. As t increases, the decrease in waiting time outweighs the increase in cost due to Figure 15(c). As $b \rightarrow 0$, the compound policy approaches NJW (for long jobs) where there is no tradeoff, and the waiting time and opportunity cost are zero.

Error Sensitivity. Figure 19 plots price, waiting time, and opportunity cost as a function of the short/long job prediction error, which is both the percentage of long jobs we mispredict as short, and short jobs we mispredict as long. Similarly, each line captures the waiting time threshold error, which is both the percentage of jobs that should wait but do not, and that do not but should. The graph shows price (a) is directly proportional to the short/long job prediction error, such that a 1% increase in error causes a 1%

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems



(a) Normalized Price (b) Mean Job Waiting Time

Fig. 20. Normalized price (a) and mean job waiting time (b) as a function of fixed resources s when executing our real production batch workload under SWW for different over-prediction errors f_{over} and NJW.

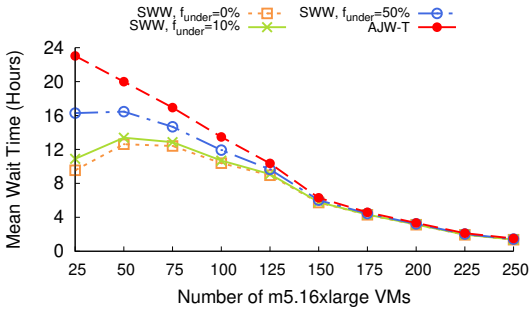


Fig. 21. Mean waiting time as a function of fixed resources s when executing our real production batch workload under SWW for different over-prediction errors f_{under}.

increase in price. In contrast, waiting time threshold errors are non-linear, with progressively lower price increases for each 10% increase in error. The graph still shows large savings compared to using on-demand even under high error rates. The mean waiting time (b) is much less affected by the short/long job prediction error, since a similar number of jobs must still wait (it is just the long jobs not waiting that increases the price). Higher values of error$_b$ actually decrease mean waiting time: while a larger percentage of (long) jobs that do not wait but should increases price, it decreases waiting time. As above, the waiting time trend dominates the opportunity cost (c), and thus shows a similar trend.

6.4 Effect of Prediction Accuracy

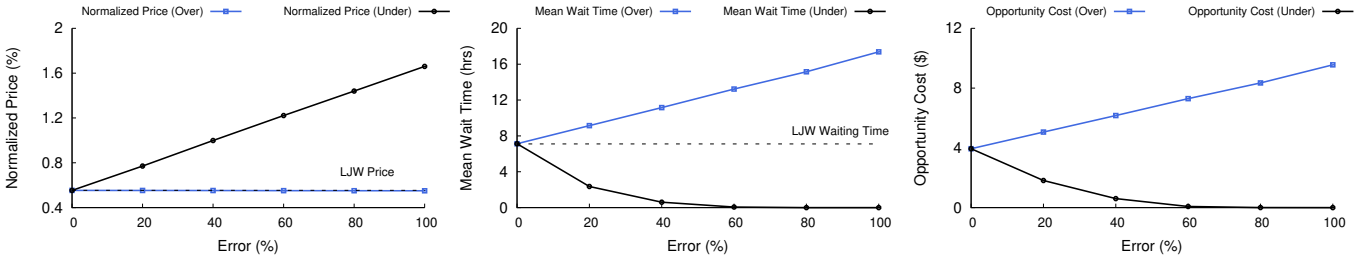
To understand the effect of prediction accuracy for our waiting policies, we vary the errors in estimating job waiting time and running time in terms of their over- and under-predictions as in our analysis from §4.1.1 and §4.2.1. We use the baseline values of b=24h for the waiting time threshold and t=3m for the long job threshold. As in our model analysis, we consider the case of over predictions and under predictions separately based on the fraction of jobs f_{over} and f_{under} that over- and under-predict their job waiting time and job running time for SWW and LJW, respectively. In particular, we simulating each waiting policy using our job trace, we explicitly control the percentage of jobs that over- and under-predict waiting times and running times. For example, if we set f_{over} for the waiting time threshold to N%, this means that N% of the jobs that would have waited (due to having a short waiting time) will now run on on-demand resources due to an over-prediction of their waiting time. We emulate this over-prediction by uniformly randomly sampling N% of jobs that should wait, and instead run them on on-demand resources. Similarly, if we set f_{under} for the waiting time threshold to M%, this means that M% of the jobs that should not have waited (due to having a long waiting time) will now wait due to an under-prediction of their waiting time. We again emulate this under-prediction by uniformly randomly sampling M% of jobs that should not wait, and instead force them to wait for fixed resources. We use the same approach to simulate over- and under-prediction errors for job running time.

Over-predicting Waiting Time. Figure 20 plots the normalized price P and mean job waiting time W as a function of fixed resources s for different prediction errors f_{over} under SWW, where f_{over} is fraction of the jobs over-predicting their waiting time and thus runs it on on-demand resources when it should have waited for fixed resources. As the graph shows, the normalized price (a) increases with the over-prediction error. As f_{over} increases, the cost of running the workload under SWW increases and approaches that of NJW. This graph mirrors Figure 7 from §4.1.1 that uses our analytical model to quantify the effect of over-predictions of waiting time on price. The only difference here is that the scheduler is work-conserving, so NJW serves as a strict upper-bound on price even when the fixed resources are over-provisioned. Figure 20(b) shows that the mean job waiting time decreases as f_{over} increases and eventually approaches 0 (or the behavior of NJW). Similar to above, this graph mirrors Figure 8 from §4.1.1 that uses our analytical model to quantify the effect of over-predictions of waiting time on the mean wait time. As in our model, the mean wait time is monotonically decreasing and approaches 0 as the number of fixed resources increases. Importantly, our empirical results on over-predictions reinforce the key points from our models in §4.1.1: that SWW is highly sensitive to over-predictions, such that 3-5% over-predictions significantly alters both the normalized price and the mean waiting time.

Under-predicting Waiting Time. Figure 21 plots the mean job waiting time w as a function of fixed resources s over different prediction errors f_{under} using SWW, where again f_{under} is fraction of the jobs that under-predict their waiting time. Thus, these under-predicting jobs wait for fixed resources when they should have run immediately on on-demand resources. As expected, the mean waiting time w increases as f_{under} increases, such that the mean waiting time approaches 0 as f_{under} increases. Since the normalized price under SWW and AJW-T remains the same regardless of the under-prediction error, we omit it here. The graph exhibits the same trend as our model predicts from Figure 9 from §4.1.1. In addition, our empirical results also emphasize the key point from our model, which is that under-predicting the waiting time does not have a significant affect on the results: it does not alter the normalized price, and it only affects the mean waiting time when the fixed resources are under-provisioned.

LJW Prediction Accuracy. Figure 22 plots the normalized price,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2021.3086270, IEEE Transactions on Parallel and Distributed Systems



(a) Normalized Price (b) Mean Job Waiting Time (c) Opportunity Cost

Fig. 22. Normalized price (a), mean job waiting time (b), and opportunity cost (c) as a function of the fraction of jobs with over- and under-prediction errors (%) in job running time for s=200 `m5.16xlarge VMs` and t=3 minutes when executing our real production batch workload under LJW.

mean waiting time, and opportunity cost under LJW policy as a function of the fraction of jobs with inaccurate runtime predictions. We plot the normalized price in (a) for both over-predictions and under-predictions. In this case, our experiment assumes that the number of fixed resources is 200 `m5.16xlarge` VMs and the long job threshold is 3 minutes. For over-predictions, the x-axis represents the fraction of jobs with runtime less than the long job threshold t where we over-predict the running time to be greater than t, while for under-predictions, the x-axis represents the fraction of jobs with runtime greater than t where we under-predict the running time to be less than t. The dotted line shows the price (a) and mean waiting time (b) from LJW with perfect predictions of job running time. As above, our empirical results match the trends shown by our analytical models in Figures 11(a) and 11(b) from §4.2.1. In particular, our results show that increasing under-prediction errors has little-to-no effect on the normalized price, but results in a linear increase in waiting time. In contrast, increasing over-prediction errors result in a linear increase in price, but a super-linear decrease in mean waiting time. Finally, Figure 22(c) shows the opportunity cost for over- and under-prediction errors as a function of the error rate. As in Figure 11(c) from §4.2.1, this graph shows that for our real workload, LJW is more sensitive to over-predictions of job running time than under-predictions, since under-predictions cause waiting time to quickly drop to zero, while over-predictions cause a significant increase in waiting time.

7 Generalization beyond Clouds

While we focus on cloud platforms in this paper, our queuing models are general and can also apply to other resources. Just as general queuing models have proven useful for decades in better understanding scheduling policies in a variety of contexts, we believe they will also prove useful in understanding waiting policies. As physical infrastructure becomes increasingly networked and programmatically driven, the cloud IaaS model is starting to spread to other sectors, such as transportation and energy, where schedulers can similarly choose between using *buying* or *renting* resources to service various types of "jobs." Under this emerging Anything-as-a-Service (XaaS) model, schedulers face the same problem as in the cloud: they must determine how many fixed resources to provision versus rent on demand based on their expected workload to optimize their cost and job waiting times. There are many emerging scenarios in other domains, including transportation and energy, where waiting policies may apply, albeit under slightly different circumstances. We discuss scenarios in energy and transportation where waiting policies may apply, and how they might differ from their use with cloud platforms. Of course, fully adapting our models to other scenarios is future work, since, similar to the cloud, each scenario presents its own specific context, which might require adaptations to the models.

Transportation. Uber, Lyft, and others have enabled on-demand transportation by connecting those needing rides with cars (and drivers) willing to provide them. As a result, users can now choose between buying their own car, or renting cars on-demand on a per-ride basis. These services have now evolved to transporting, not only people, but also packages within urban environments, e.g., Uber Connect and Lyft Essential Deliveries. Thus, similar to cloud schedulers, companies that schedule package deliveries now have a choice between maintaining their own vehicle fleet (and drivers) to deliver packages, or renting vehicles on-demand. As with our cloud example, buying a vehicle is cheaper than renting one if its utilization is high, and thus the optimal provisioning of vehicles depends on how long the company, and its customers, are willing to wait for their packages. In this case, the resource is the vehicle, and the "job" is delivering the package, which takes different lengths of time depending on the distance to the destination. Unlike with computing, the distance and thus the job length is well-known, although there may be some variation due to traffic. With the emergence of autonomous vehicles, we expect this scenario to become even more important.

Energy. A similar concept also occurs in the energy sector when considering choosing between using locally generated solar energy and grid energy. In this case, solar energy is fixed, since it requires an initial capital expense, while grid energy is "rented" on-demand since users can access it anytime and pay only for what they use. Solar energy's cost is lower than grid energy is most locations, as long as it can be productively used. We assume here that locally generated solar energy is not "grid-tied," and thus cannot be sold back to the grid. Such grid-tied solar is increasingly being restricted as grid solar penetration increases, requiring users to consume their excess solar energy locally or store it in a battery. Waiting policies apply to this scenario, since there is a choice between using locally generated solar energy and grid energy to execute some "job." For example, consider a solar-powered EV charging station where each "job" is the task of fully charging an EV. An EV charging scheduler must decide whether fixed solar energy or on-demand grid power should charge each EV. The optimal provisioning of solar capacity depends on how long the charging station is willing to have users wait for solar energy to become available. However, a key difference from the cloud example is that the fixed solar resource's capacity is variable in addition to job durations and inter-arrival times. Thus, optimal provisioning for solar requires extending our model to account for stochasticity in the fixed resource capacity, which is part of future work. Even so, waiting policies that which jobs wait, and for how long, are fundamental to optimally provisioning these systems.

8 Related Work

Our work is related to, and builds on, a wide variety of prior work in multiple different areas, which we summarize below.

Cloud Computing. While our queuing model is general and applies to any XaaS scheduler, our primary motivating example is from cloud platforms that present users with a choice of renting cloud resources on demand at a higher price than purchasing them (or reserving them for a long period). While prior work focuses on optimizing the provisioning of reserved VMs in the cloud, it makes simple workload assumptions. In particular, prior work often assumes the workload is continuous and uniform, rather than composed of discrete jobs, which leads to solutions based on dynamic and integer programming [15], [23], [24], [30], [33], [34]. The canonical application is a distributed web server with a front-end load balancer that distributes requests. As a result, this work does not explore the tradeoff between price and job waiting time. These techniques do not map to cloud-enabled job schedulers, such as Kubernetes and Slurm, that must schedule jobs on on-demand and fixed resources.

Shen et al. focus on a similar problem in the context of a job scheduler, but do not permit any queuing, and instead use integer programming to determine the size of VMs to allocate and how to efficiently pack jobs onto on-demand and reserved VMs [30]. Our queuing models do not consider jobs with different resource requirements, and how to bin pack them on resources. Since providers typically offer VMs in fixed sizes, this results in some model inaccuracy and cost inefficiency.

Queuing Theory and Marginal Analysis. Our work applies a number of previously developed queuing theory results to gain insights into key tradeoffs exposed by different waiting policies. In particular, our work builds on classic marginal analysis and queuing results by Erlang and others [19], [25], [31], [35]. The emergence of cloud-enabled schedulers is increasing the importance of these results in optimizing cost. As we discuss, AJW's analysis is simply that of an $M/M/s/\infty$ queue, and NJW's analysis applies classic marginal analysis where jobs never wait for resources [25]. Our analysis for AJW-T and SWW combines recent results on reneging and balking by Liu and Kulkarni [27] with classic marginal analysis, and shows how a waiting time threshold defines a spectrum between AJW and NJW. Our SWW analysis also models inaccurate waiting time predictions.

In general, reneging and balking are examples of "customer abandonment" policies from queuing theory, which model customers, i.e., jobs, becoming impatient and leaving the queue. Many of these models are probabilistic and assume an increasing fraction of customers (or jobs) abandon the queue as their waiting time increases based on diverse customer preferences. These customer-centric models do not apply to our context, where the waiting policy determines whether jobs abandon the queue (and run on on-demand resources). Finally, our LJW analysis leverages a well-known approximation for the waiting time of a $M/G/s/\infty$ queue. Finally, our compound policy analysis combines and extends elements from each waiting policy.

Ski Rental Problems. Our problem is similar to the classic ski rental problem in online algorithms [11]. However, the assumption in online algorithms is that there is no (or limited) knowledge of the future, whereas our queueing analysis leverages a workload characterization to model the system. Ski rental problems also typically focus on whether to buy or rent a single resource whereas our problem focuses on provisioning, i.e., how many resources to buy versus rent, and generally do not consider the cost and waiting time tradeoff. Recent work examines improving online algorithms, including the ski rental problem and job scheduling, using ML predictions [26]. Our model accounts for inaccurate predictions of waiting time made by ML classifiers, and thus offers a benchmark accuracy that ML classifiers must satisfy to achieve specific waiting time and cost targets.

Job Scheduling. Our work is orthogonal to prior work on job scheduling for fixed resources. A waiting policy is the dual of a scheduling policy: while a scheduling policy determines which jobs should execute when fixed resources are available, a waiting policy determines which jobs should wait when fixed resources are not available. Given a cloud platform, where jobs never need wait, the waiting time and cost to execute jobs is a function of the waiting policy. The scheduling policy may also affect waiting time and cost. Our simple models assume FCFS scheduling. Some our waiting policies exhibit similar properties as scheduling policies. For example, LJW is akin to shortest job first scheduling, and reduces mean waiting time for a modest increase in cost.

Prediction Accuracy. The SWW, LJW, and compound waiting policies require knowledge of job runtime and queue waiting times, which, as we discuss, may not be available *a priori*. There is significant prior work on methods for predicting both job running time and queue waiting time [16], [17], [20], [29], [32]. For example, [32] estimates job runtime by categorizing jobs using their common attributes, such as user ID or resources requested, and chooses the estimate from the category that has produced the best estimates in the past. Similarly, [29] presents a simple job runtime and waiting time prediction model for a fixed cluster (or grid) system, while [17] develops a model to derive the upper bound of a job's duration based on both workload and cluster load prediction errors. [18] also utilize runtime predictions to derive an upper bound on the cost required to execute a workload, assuming a particular margin in their prediction errors. Our goal in this paper is not to improve upon this prior work, but to highlight the asymmetry in over- and under-predictions with respect to waiting policies, which can enable future work on prediction methods to better contextualize their accuracy for cloud-enabled schedulers.

9 Conclusion

This paper introduces the concept of a waiting policy for cloud-enabled schedulers, and defines, models, analyzes, and empirically validates multiple fundamental waiting policies. Our analysis reveals key tradeoffs in designing waiting policies under FCFS and SJF scheduling, and also captures the impact of inaccurate predictions of job running time and waiting time on the fixed resource provisioning, price, and mean waiting time. A goal of this paper is to provide a formal foundation for future work on waiting policies both analytically and empirically, including on more general distributions of job inter-arrival and service times, different scheduling policies, and machine learning (ML) classifiers to accurately estimate job waiting and running times. In addition, waiting policies are important in understanding how users value and provision fixed and on-demand resources. Understanding these user valuations is important for cloud providers in determining how to set the price of fixed and on-demand resources to maximize their revenue. Finally, while our paper focuses on evaluating waiting policies in the context of cloud platforms, as we discuss in §7, the concept is general and may also apply to emerging XaaS-enabled schedulers for other resources.

REFERENCES

[1] Kubernetes on AWS. https://kubernetes-incubator.github.io/kube-aws/, Accessed May 2018.

[2] Google Kubernetes Engine. https://cloud.google.com/kubernetes-engine/, Accessed October 2019.

[3] Slurm Elastic Computing (Cloud Bursting). https://slurm.schedmd.com/elastic_computing.html, Accessed October 2019.

[4] Slurm Workload Manager. https://slurm.schedmd.com/, Accessed October 2019.

[5] AWS OutPost. https://aws.amazon.com/outposts/, Accessed August 2020.

[6] AWS ParallelCluster Auto Scaling. https://docs.aws.amazon.com/parallelcluster/latest/ug/autoscaling.html, Accessed April 2020.

[7] UMass Trace Repository. http://traces.cs.umass.edu/, Accessed August 2020.

[8] University of Massachusetts Green High Performance Computing Cluster. http://wiki.umassrc.org/wiki/index.php/Main_Page, Accessed August 2020.

[9] Waiting Game Job Simulator. https://doi.org/10.5281/zenodo.3875634, Accessed August 2020.

[10] Waiting Game Job Trace. https://doi.org/10.5281/zenodo.3872168, Accessed August 2020.

[11] L. Ai, X. Wu, L. Huang, L. Huang, P. Tang, and J. Li. The Multi-shop Ski Rental Problem. In *SIGMETRICS*, June 2014.

[12] P. Ambati, N. Bashir, D. Irwin, and P. Shenoy. Waiting Game: Optimally Provisioning Fixed Resources for Cloud-Enabled Schedulers. In *SC*, November 2020.

[13] J. Brodkin. ArsTechnica, Netflix finishes its massive migration to the Amazon cloud. https://arstechnica.com/information-technology/2016/02/netflix-finishes-its-massive-migration-to-the-amazon-cloud/, February 11th 2016.

[14] J. Chen. Medium, Why building your own Deep Learning Computer is 10x cheaper than AWS. https://medium.com/the-mission/why-building-your-own-deep-learning-computer-is-10x-cheaper-than-aws-b1c91b55ce8c, September 24th 2018.

[15] R. V. den Bossche, K. Vanmechelen, and J. Broeckhove. IaaS Reserved Contract Procurement Optimisation with Load Prediction. *Future Generation Computer Systems*, 53, December 2015.

[16] S. Di, D. Kondo, and C. Wang. Optimization and Stabilization of Composite Service Processing in a Cloud System. In *2013 IEEE/ACM 21st International Symposium on Quality of Service (IWQoS)*, June 2013.

[17] S. Di, C. Wang, and F. Cappello. Adaptive Algorithm for Minimizing Cloud Task Length with Prediction Errors. *IEEE Transactions on Cloud Computing*, 2(2):194–207, 2014.

[18] S. Di, C. Wang, D. Kondo, and G. Han. Towards Payment-Bound Analysis in Cloud Systems with Task-Prediction Errors. In *2013 IEEE Sixth International Conference on Cloud Computing*, June 2013.

[19] A. K. Erlang. *On the Rational Determination of the Number of Circuits (1924)*. In Life and Works of A K. Erlang , E. Brockmeyer, H J. Halstrom and A. Jensen, Danish Academy of Technical Science, 1948.

[20] R. Grandl, M. Chowdhury, A. Akella, and G. Ananthanarayanan. Altruistic Scheduling in Multi-Resource Clusters. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, 2016.

[21] T. Guo, U. Sharma, S. Sahu, T. Wood, and P. Shenoy. Seagull: Intelligent Cloud Bursting for Enterprise Applications. In *USENIX ATC*, June 2012.

[22] T. Hoff. High Scalability, The Eternal Cost Savings of Netflix's Internal Spot Market. http://highscalability.com/blog/2017/12/4/the-eternal-cost-savings-of-netflixs-internal-spot-market.html, December 4th 2017.

[23] Y. Hong, J. Xue, and M. Thottethodi. Dynamic Server Provisioning to Minimize Cost in an IaaS Cloud. In *SIGMETRICS*, June 2011.

[24] M. Hu, J. Luo, and B. Veeravalli. Optimal Provisioning for Scheduling Divisible Loads with Reserved Cloud Resources. In *ICON*, December 2012.

[25] A. Jensen. *Moe's Principle: An Econometric Investigation Intended as an Aid in Dimensioning and Managing Telephone Plant*. The Copenhagen Telephone Company, 1950.

[26] R. Kumar, M. Purohit, and Z. Svitkina. Improving Online Algorithms via ML Predictions. In *NIPS*, December 2018.

[27] L. Liu and V. Kulkarni. Balking and Reneging in M/G/s Systems: Exact Analysis and Approximations. *Probability in the Engineering and Informational Sciences*, 22(3), July 2008.

[28] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen. Cost-effective Cloud HPC Resource Provisioning by Building Semi-Elastic Virtual Clusters. In *SC*, November 2013.

[29] S. Omer, N.Yigitbasi, A. Iosup, and D. Epema. Trace-based Evaluation of Job Runtime and Queue Wait Time Predictions in Grids. In *HPDC*, June 2009.

[30] S. Shen, K. Deng, A. Iosup, and D. Epema. Scheduling Jobs in the Cloud using On-demand and Reserved Instances. In *Euro-Par*, August 2013.

[31] L. Takacs. *Introduction to the Theory of Queues*. Oxford University Press, 1962.

[32] A. Tumanov, A. Jiang, J. Park, M. Kozuch, and G. Ganger. Jamaisvu: Robust Scheduling with Auto-Estimated Job Runtimes, Accessed September 2016.

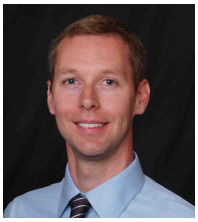
[33] W. Wang, B. Li, and B. Liang. To Reserve or Not to Reserve: Optimal Online Multi-Instance Aquisition in IaaS Clouds. In *ICAC*, June 2013.

[34] W. Wang, D. Niu, B. Li, and B. Liang. Dynamic Cloud Resource Reservation via Cloud Brokerage. In *ICDCS*, July 2013.

[35] W. Whitt. Erlang B and C Formulas: Problems and Solutions. http://www.columbia.edu/ ww2040/ErlangBandCFormulas.pdf, 2002.

Pradeep Ambati Pradeep Ambati is a Ph.D. candidate in the Electrical and Computer Engineering department at the University of Massachusetts Amherst. He received his M.S. in Electrical and Computer Engineering from the University of Massachusetts Amherst in 2017, and his B.E. in Electrical, Electronics, and Communications Engineering at Chaitanya Bharathi Institute of Technology in 2012. His research interests are in optimizing the management of cloud platforms.

Noman Bashir Noman Bashir is Ph.D. candidate in the Electrical and Computer Engineering department at the University of Massachusetts Amherst. He received his M.S. in Energy Systems Engineering from the National University of Science and Technology, Islamabad in 2016, and his B.S. in Electrical Engineering at the University of Engineering and Technology, Lahore in 2013. His research interests are in data-driven, machine learning techniques to optimize the performance of distributed systems.



David Irwin David Irwin is an Associate Professor in the Department of Electrical and Computer Engineering at the University of Massachusetts Amherst. He received his Ph.D. and M.S. in Computer Science from Duke University in 2007 and 2005, respectively, and his B.S. in Computer Science and Mathematics from Vanderbilt University in 2001. His research interests are broadly in experimental computing systems with a particular emphasis on sustainability.

Prashant Shenoy Prashant Shenoy received the B.Tech degree in Computer Science and Engineering from the Indian Institute of Technology, Bombay, in 1993, and the M.S and Ph.D. degrees in Computer Science from the University of Texas, Austin, in 1994 and 1998, respectively. He is currently a Distinguished Professor of Computer Science at the University of Massachusetts. His current research focuses on distributed systems and networking. He is a fellow of the ACM, the IEEE, and the AAAS.