Waiting Game: Optimally Provisioning Fixed Resources for Cloud-enabled Schedulers

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

***Abstract*—While cloud platforms enable users to rent computing resources on demand to execute their jobs, buying fixed resources is still much cheaper than renting if their utilization is high. Thus, optimizing cloud costs requires users to determine how many fixed resources to buy versus rent based on their workload. In this paper, we introduce the concept of a *waiting policy* for cloud-enabled schedulers, which is the dual of a scheduling policy, and show that the optimal cost depends on it. We define multiple waiting policies and develop simple analytical models to reveal their tradeoff between fixed resource provisioning, cost, and job waiting time. We evaluate the impact of these waiting policies on a year-long production batch workload consisting of 14M jobs run on a 14.3k-core cluster, and show that a compound waiting policy decreases the cost (by 5%) and mean job waiting time (by 7×) compared to a fixed cluster of the current size.**

I. Introduction

Cloud platforms enable users to rent computing resources on demand, in the form of virtual machines (VMs), to execute their jobs. 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 each job, 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 Amazon Web Services (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 [17], [24], as do many companies, which both buy reserved VMs and dynamically rent on-demand VMs [18].

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. A waiting policy is the dual of a scheduling policy: 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 important, since it dictates the tradeoff between job performance and cost. Waiting policies also differ from auto-scaling policies used by cloud-enabled schedulers, which implicitly assume jobs should never wait and immediately acquire resources to satisfy queued jobs [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 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. Our

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 I
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.

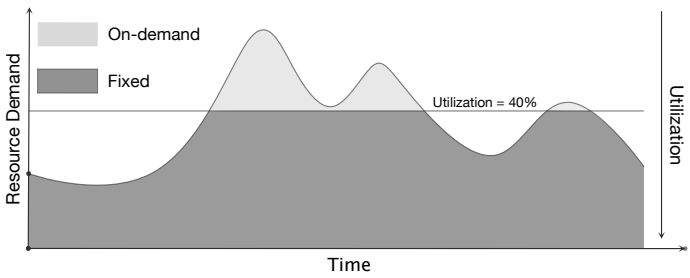


Fig. 1. Illustration of utilization for each unit of stacked resource demand and the break even point at 40% utilization.

hypothesis is that, by optimizing their waiting policy, cloud-enabled schedulers can significantly 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.

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 cloud 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 cloud cost. Our modeling and analysis also provides the necessary formal foundation for conducting any future work on waiting policies.

Implementation and Evaluation. We implement our waiting policies in a trace-driven job simulator, and evaluate their impact on a real year-long batch 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×) 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).

II. 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 assume a cloud platform that offers two types of VMs: on-demand and fixed. Users may acquire and release on-demand VMs any time, and pay only for the time they use them without any commitment. In contrast, users must commit to paying for fixed VMs over a long period, e.g., one or more years. Importantly, however, fixed VMs are cheaper than on-demand VMs if they are highly utilized. While cloud platforms also offer spot [5], [7] or preemptible VMs [8], which are often cheaper than highly utilized fixed VMs, not all jobs can use them. We discuss spot VMs further in §IV-D.

Table I 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 §I, 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 cloud platform, 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, cloud 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," also contribute to lowering cloud platforms' effective price for fixed resources. Even so, as our example illustrates, highly utilized fixed resources are still much cheaper, since they eliminate the cloud 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 VMs to execute a workload on a cloud platform to minimize cost is a classic marginal analysis problem [21]. 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 I. 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 sth 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 [16], [27], [30]. Variants of this classic problem have been addressed in prior work both generally, and in the context of cloud computing, which we discuss in §VII.

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. We know of no work that explicitly defines and analyzes such waiting policies for cloud-enabled schedulers by applying marginal analysis.

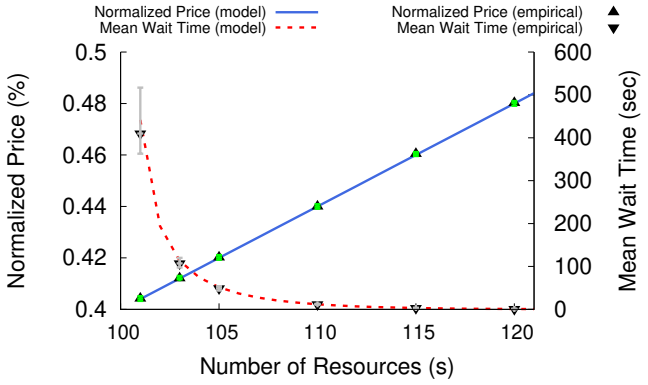


Fig. 2. Normalized price P (left y-axis) and mean wait time w (right y-axis) as a function of fixed resources s under AJW. Mean wait time $w{\to}\infty$ as fixed resources $s{\to}100$, and mean wait time $w{\to}0$ as fixed resources $s{\to}\infty$.

III. 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. 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 §IV, 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 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 utilization) per fixed resource is $\rho{=}a/s{=}\lambda/(s{\times}\mu)$.

A. 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 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 §III and §IV 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 §II, 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 §V. Each trial simulates the model on a synthetically generated job trace with 2M jobs using exponentially distributed inter-arrival and service times using 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 2M jobs, there is almost no deviation from the average on each trial.

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 (when 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.

B. 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

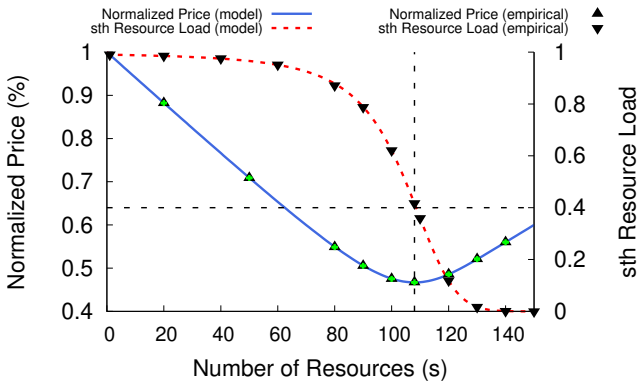


Fig. 3. Normalized price P (left y-axis) and mean utilization of the s^{th} resource ρ_s (right y-axis) as a function of fixed resources s under NJW. The minimum price occurs when the fixed resources' discount factor d=ρ_s.

when all fixed resources are busy. Recall from §II 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.

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 s^{th} 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 s^{th} 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%

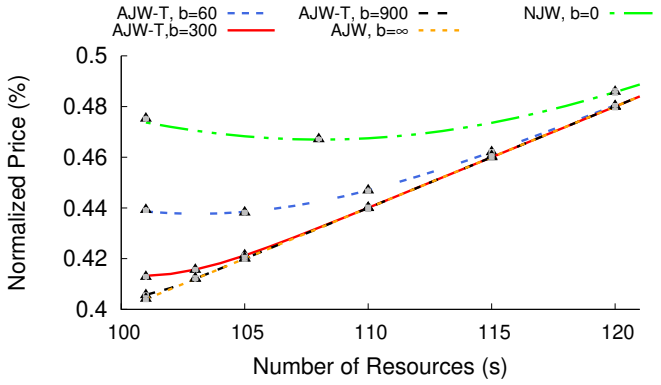


Fig. 4. Normalized price P as a function of fixed resources s under AJW-T for different threshold waiting times b.

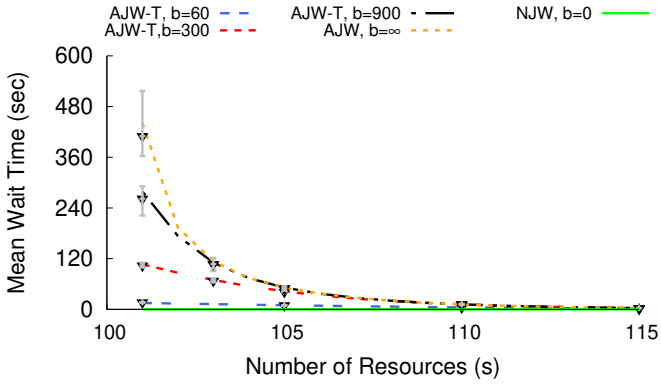


Fig. 5. Mean waiting time w as a function of fixed resources s under AJW-T for different threshold waiting times b.

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.

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

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.

Empirical Validation. We empirically validate NJW using the same baseline example from §III-A. 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 s^{th} 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 ∼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.*

C. 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 [23].

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

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

where

$$\delta = (s\mu - \lambda) \quad (7)$$

$$\beta = \frac{s\mu p}{1-p} \quad (8)$$

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

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

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 $a{=}\lambda/\mu$.

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

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 [23]. 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.

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

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 $s{\rightarrow}100$. Thus, unlike AJW and NJW, AJW-T is configurable, enabling users to set their own 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.*

IV. 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.

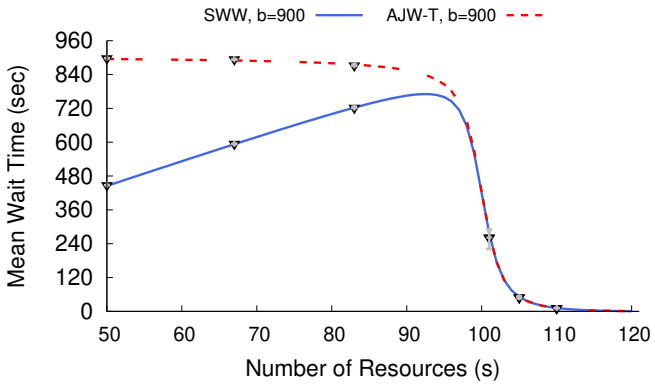


Fig. 6. Mean waiting time as a function of fixed resources under SWW and AJW-T where b=900s=15m.

A. 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 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 [23]. 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.

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

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 is strictly better than AJW-T for the same threshold b, yielding same price at a lower mean waiting time.*

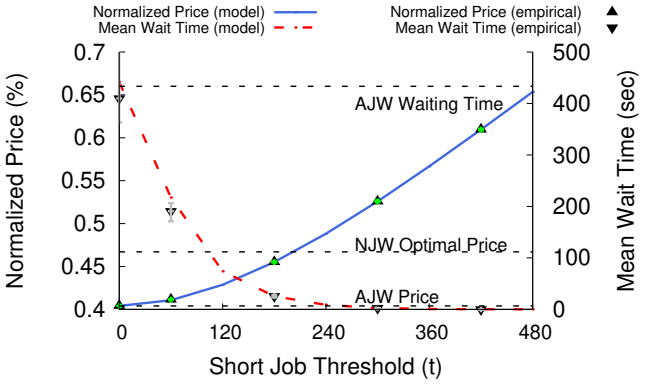


Fig. 7. Normalized price P and mean wait time w as a function of the short job threshold t (in seconds) for s=101 under an LJW waiting policy.

B. 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.

$$P = (1-r) \times \frac{\mu}{\mu_{long}} \times P_{long} + r \times \frac{\mu}{\mu_{short}} \times P_{short} \quad (14)$$

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.

$$w = (1-r) \times w_{long} \quad (15)$$

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.

$$r = 1 - e^{-\mu t} \quad (16)$$

Long Jobs. Since long jobs always wait for fixed resources, the policy is similar to AJW in §III-A 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 $(1-r)$.

$$\lambda_{long} = \lambda \times (1-r) = \lambda \times e^{-\mu t} \quad (17)$$

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.

$$f(x,\mu) = \mu e^{-\mu(x-t)}, x \geq t \quad (18)$$

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→∞.

$$\frac{1}{\mu_{long}} = \int_t^{\infty} x\mu e^{-\mu(x-t)} dx = t + \frac{1}{\mu} \quad (19)$$

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.

$$\frac{1}{\mu} = r \times \frac{1}{\mu_{short}} + (1-r) \times \frac{1}{\mu_{long}} \quad (20)$$

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 $\rho_{long} = \lambda_{long}/(s \times \mu_{long})$.

$$P_{long} = \frac{p_f}{\rho_f} = \frac{p_f \times s \times \mu_{long}}{\lambda_{long}} \quad (21)$$

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.

$$w \sim \frac{CV^2+1}{2} \times \frac{C(s,a)}{s \times \mu_{long} - \lambda_{long}} \quad (22)$$

Empirical Validation. Figure 7 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

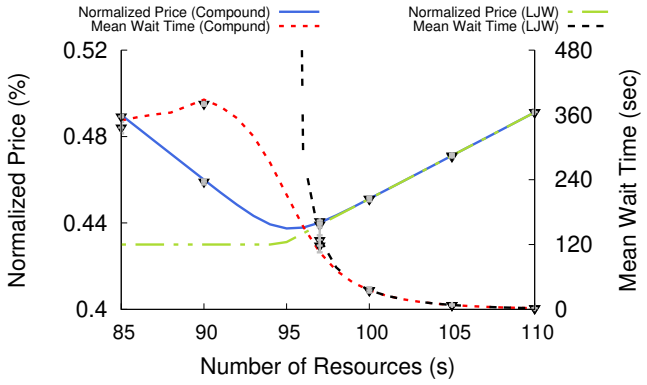


Fig. 8. Normalized price P and mean wait time w as a function of fixed resources s for our compound policy (b=900 and t=180) and LJW (t=180).

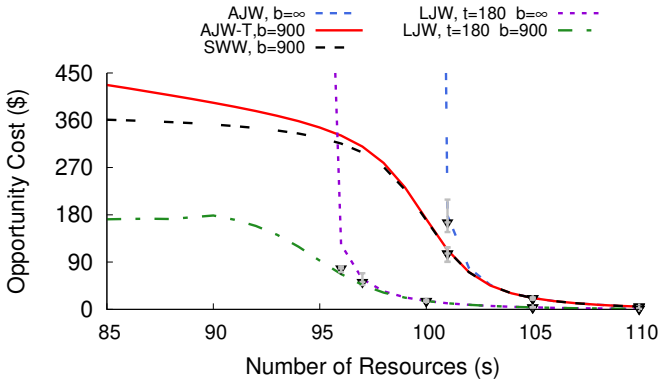


Fig. 9. Opportunity cost as a function of fixed resources s under AJW, AJW-T, SWW, LJW, and compound policy.

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.*

C. 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 §IV-B, 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 §IV-A 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 §IV-B 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.

$$P = (1-r_{short}) \times (1-r_{sww}) \times \frac{\mu}{\mu_{long}} \times \frac{p_f}{\rho_f} + (1-r_{short}) \times r_{sww} \times \frac{\mu}{\mu_{long}} \times p_o + r_{short} \times \frac{\mu}{\mu_{short}} \times p_o \quad (23)$$

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.

$$\rho_f = \frac{(1-r_{short}) \times (1-r_{sww}) \times \lambda}{s \times \mu_{long}} \quad (24)$$

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.

$$w \sim \begin{cases} \frac{CV^2+1}{2} \times (1-r_{sww}) \times \left(\frac{\alpha \times \beta (1-\delta b e^{-\delta \times b} - e^{-\delta \times b})}{(1-r_{sww}) \times \delta^2}\right) & \rho < 1 \\ \frac{CV^2+1}{2} \times (1-r_{sww}) \times \left(\frac{\alpha \times \beta \times b^2}{(1-r_{sww}) \times 2}\right) & \rho = 1 \end{cases} \quad (25)$$

Empirical Validation. Figure 8 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.*

D. 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 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 $P \times w$ and is in dollars, where lower values of P and w are better. Figure 9 shows the mean opportunity cost of 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. As shown, for the remaining policies where a price-waiting time tradeoff exists, our compound policy yields the lowest opportunity cost.

We can trivially extend our average-case analysis to spot (or preemptible) VMs. Spot VM prices p_s are typically 10-20% of the on-demand price p_o, and 25%-50% of the fixed reserved price p_f at full utilization, even after considering the increase in running time and cost due to revocations [22], [25]. Thus, on average, spot VMs are always cheaper than on-demand and fixed VMs, and preferable for jobs without deadlines. As a result, any job that can run on spot VMs should run on them. However, not all jobs can handle revocations, which are akin to failures. For example, many jobs are not idempotent and cannot restart after a revocation. Given this, we can extend our average-case analysis to spot VMs by simply adjusting the workload to remove spot-compatible jobs that can run on spot VMs. We then apply the same analysis above on the remaining workload, and adjust the effective price and waiting time to account for the fraction of workload that runs on spot VMs.

Finally, 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. 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 §VI echoes this point by showing that the relative price, waiting time, and opportunity cost between the waiting policies of a real production workload precisely follows our analysis.

V. 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 §III and §IV.

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.

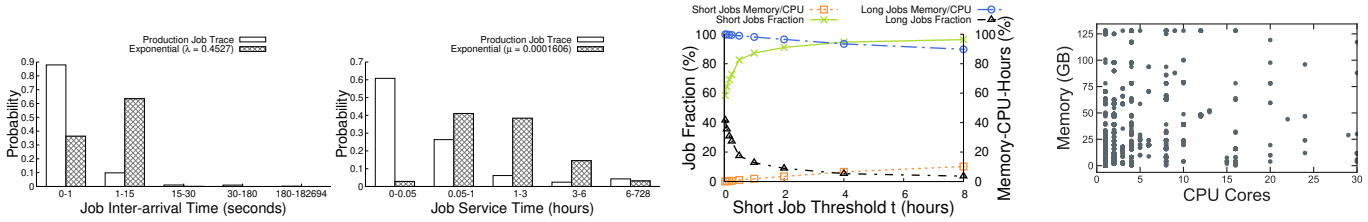
Each job in the input trace has a service time based on its resource, e.g., core and memory, requirements. Our job simulator packs jobs onto fixed resources using a simple best-fit heuristic, and, when scheduling a job on an on-demand VM, always selects the smallest one that satisfies the job's resource requirements. We have publicly released our job simulator at the UMass Trace Repository [9], [11].

Real-world Data. In §VI, 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, including its medical school [10]. Thus, the workload is a representative sample of job types across the entire scientific, engineering, and medical research communities. The cluster is located at the Massachusetts Green High Performance Computing Center (MGHPCC), a 15MW data center in Holyoke, Massachusetts that also hosts computing 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. We modify the raw trace to conform to our job simulator's input format. We have publicly released this job trace at the UMass Trace Repository as a basis for further research [9], [12].

VI. 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.

Workload. Figure 10 characterizes our real workload and our model's ideal. Figure 10(a) is a histogram of job inter-arrival



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

Fig. 10. 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).

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 10(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 10(c) shows the fraction of long and short jobs, and their resource usage (in memory×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 10(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 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.

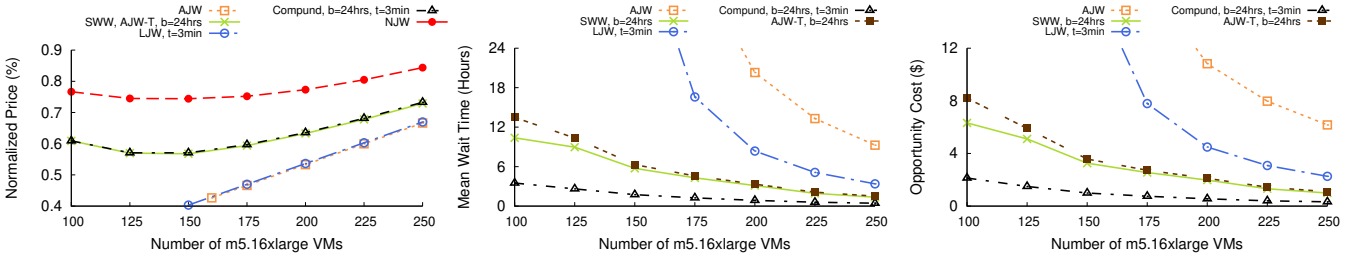
A. Real-world Workload Results

Figure 11 shows the normalized price (a), mean waiting time (b), and opportunity cost (c) for each of our waiting policies. 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 11(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 11(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



(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 `m5.16xlarge` VMs when executing our real production batch workload under AJW, AJW-T, SWW, LJW, and our compound policy.

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 11(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 9.

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).*

B. 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, as noted above.

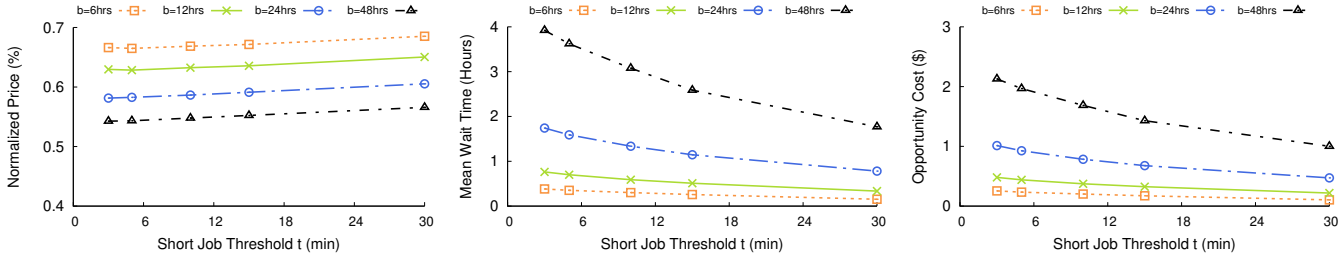
Parameter Sensitivity. Figure 12 plots price, waiting time, and opportunity cost as a function of the short job threshold t with lines for different values of the waiting time threshold b. We vary t from 3-30m and the waiting time threshold from 6h-48h. The price (a) increases linearly with the short 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 10(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 10(c). As $b \to 0$, 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 13 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% 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. Finally, as above, the waiting time trend dominates the opportunity cost (c), and thus shows a similar trend.

VII. Related Work

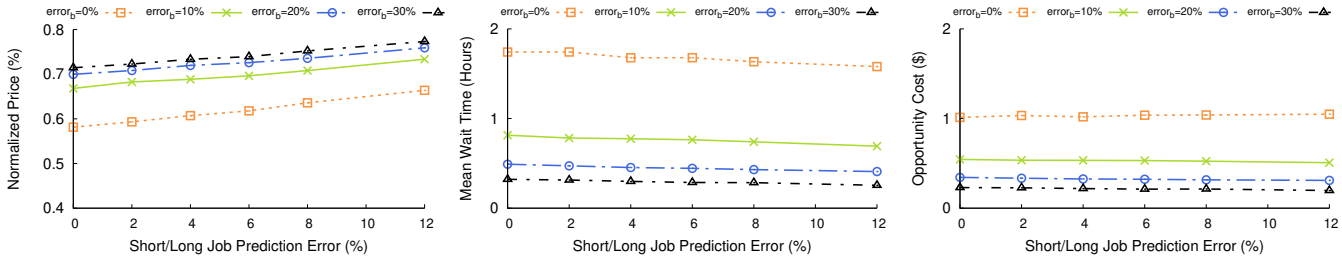
Our work is related to prior work in many different areas.

Cloud Computing. While some prior work focuses on optimizing the provisioning of reserved cloud VMs, 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], [19], [20], [26], [28], [29]. The canonical application is a distributed web server with a front-end load balancer that distributes requests. Thus, this work does not apply to cloud-enabled job schedulers. Our work is also related to prior work on job scheduling for hybrid



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

Fig. 12. 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. 13. 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`.

clouds that run jobs on fixed resources but can also burst into the cloud [17], [24]. While hybrid cloud provisioning and scheduling is well-studied, we know of no work that focuses on explicit waiting policies. As mentioned earlier, existing auto-scaling policies for cloud-enabled job schedulers [6] define an implicit waiting policy that is equivalent to NJW.

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 [16], [21], [27], [30], as well prior results on reneging and balking [23]. For example, 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 [21]. Our analysis for AJW-T and SWW then combines recent results on reneging and balking by Liu and Kulkarni [23] with classic marginal analysis, and shows how a waiting time threshold defines a spectrum between AJW and NJW. 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).

Ski Rental Problems. Our problem is similar to the classic ski rental problem in online algorithms [13]. However, these assume there is no knowledge of the future, whereas our queueing analysis leverages a workload characterization. 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.

VIII. 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. A key 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. Specifically, our evaluation shows that real workload characteristics differ from our model's assumptions, which motivates analytical models based on more general distributions of inter-arrival and service times. We also only analyze waiting policies in conjunction with FCFS scheduling, which motivates future work on analyzing other scheduling policies, such as shortest job first. Our real-world validation is also a best case scenario, since it assumes accurate predictions of job waiting and running times. Thus, we plan to evaluate the accuracy of ML classifiers in making these predictions to determine the benefits of waiting policies in practice. We plan to implement and evaluate these ML-based waiting policies in a cloud-enabled job scheduler, such as Slurm [4].

Acknowledgements. This work is funded by National Science Foundation grants CNS-1802523 and CNS-1908536. We also thank the Research Computing team at the UMass Medical School for providing access to batch traces from the UMass Green High Performance Computing Cluster (GHPCC) [10].

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] Amazon EC2 Spot Instances. https://aws.amazon.com/ec2/spot/, Accessed June 2020.

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

[7] Azure Spot Virtual Machines. https://azure.microsoft.com/en-us/pricing/spot/, Accessed June 2020.

[8] Google Preemptible Virtual Machines. https://cloud.google.com/preemptible-vms, Accessed June 2020.

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

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

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

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

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

[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] 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.

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

[18] 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.

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

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

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

[22] J. Kadupitige, V. Jadhao, and P. Sharma. Modeling the Temporally Constrained Preemptions of Transient Cloud VMs. In *HPDC*, June 2020.

[23] 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.

[24] 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.

[25] S. Shastri, A. Rizk, and D. Irwin. Transient Guarantees: Maximizing the Value of Idle Cloud Capacity. In *SC*, November 2016.

[26] 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.

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

[28] 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.

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

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