Cost-aware Scheduling for Heterogeneous Enterprise Machines (CASH'EM)

Jennifer Burge, Partha Ranganathan, Janet L. Wiener
Enterprise Systems and Software Laboratory
HP Laboratories Palo Alto
HPL-2007-63
April 17, 2007*

power scheduling, data centers, autonomic computing

Data centers contain heterogeneous sets of machines. Some machines are faster and some – often the same ones – consume more energy and cost more to operate. The data center coordinator must decide how to allocate these machines to multiple applications of potentially many customers, each of which has different requirements. Given a stream of customer requests for machines, how does the data center provider decide which machines to give to whom and when?

We propose new algorithms for a cost-aware provider to maximize its profit as it makes admission and scheduling decisions for the customer requests. We show that it matters which machines are assigned to each customer, especially when the data center is undersaturated. (Most data centers are.) Our new algorithms do best when they try to anticipate the ”riskiness” of their decisions, that is, the likelihood that even higher-value requests will arrive later. We also show that turning unused machines off, rather than leaving them idle, even using simple heuristics like “turn off a machine that has been idle for ten minutes,” can save a lot of money. Finally, we show that having heterogeneity in the data center is, in fact, beneficial. We demonstrate that the same set of customers can be satisfied at a lower cost and a higher profit in a heterogeneous data center rather than in a data center comprised solely of the newest, fastest, machines.

* Internal Accession Date Only
Approved for External Publication

Cost-aware Scheduling for Heterogeneous Enterprise Machines (CASH'EM)

Jennifer Burge, Partha Ranganathan, Janet L. Wiener
Duke University and HP Laboratories
jen@cs.duke.edu, partha.ranganathan@hp.com, janet.wiener@hp.com

Data centers contain heterogeneous sets of machines. Some machines are faster and some – often the same ones – consume more energy and cost more to operate. The data center coordinator must decide how to allocate these machines to multiple applications of potentially many customers, each of which has different requirements. Given a stream of customer requests for machines, how does the data center provider decide which machines to give to whom and when?

We propose new algorithms for a cost-aware provider to maximize its profit as it makes admission and scheduling decisions for the customer requests. We show that it matters which machines are assigned to each customer, especially when the data center is undersaturated. (Most data centers are.) Our new algorithms do best when they try to anticipate the "riskiness" of their decisions, that is, the likelihood that even higher-value requests will arrive later. We also show that turning unused machines off, rather than leaving them idle, even using simple heuristics like "turn off a machine that has been idle for ten minutes," can save a lot of money. Finally, we show that having heterogeneity in the data center is, in fact, beneficial. We demonstrate that the same set of customers can be satisfied at a lower cost and a higher profit in a heterogeneous data center rather than in a data center comprised solely of the newest, fastest, machines.

[version of: 2006-09-19 20:51]

1 Introduction

Data centers today provide machines (and other resources, such as storage) to customers. These customers may be different departments in the same organization sharing a company-owned data center, or they may be independent entities renting time from an infrastructure server provider in a service-oriented architecture. Either way, the data center must understand its costs in order to charge its customers appropriately for time on its machines.

Costs include both provisioning costs, such as buying server and cooling hardware, and operating costs, including the energy cost to power the machines and cool them. Additional costs include those for floor space, people, and software[4]. While no one cost component dominates, energy costs are increasing – Google reports that they are over 10% of their operating costs[3] and Moore estimates that a 30,000 square foot data center can spend $4-8 million per year on electricity just to power its servers[14]. Furthermore, energy costs roughly double when the additional energy to cool the servers is included[4] – and restricting energy costs allows the data center to spend less money on cooling hardware. Conserving energy is therefore an important goal to contain costs.

However, conserving energy is not simple. Data centers nearly always have a heterogeneous pool of machines: different generation servers, different kinds of blade servers, or even a mix of blades and servers. Different machines provide different performance and consume different amounts of power. It is even possible to get different power-performance ratios for the same machine by modifying its power-states[1] or by running multiple virtual machines, each with its own power-performance ratio.

To complicate matters further, different customers offer different amounts of money and have different requirements: how fast their requests must be met, what fraction of requests must be satisfied, what penalties exist for non-compliance, etc., all of which are specified in contracts called service-level agreements (SLAs).

In this paper, we try to maximize the *profit* of the data center. Profit is a function of both value to customers and costs to satisfy them. While much prior work has looked at individually minimizing cost[9, 19] or maximizing value[7, 10], we realize that neither cost nor value is a static function and it is important to consider both[16, 2]. By understanding and comparing the power-performance ratios of the different machines to the SLAs of the customers, the data center provider can make informed choices about which customers get which machines when.

The data center provider must make two key kinds of decisions. First, it must make both initial and incremental provisioning decisions. Which machines to buy? Which machines to replace? Second, it must make scheduling decisions: Which workloads (sets of customers) are best to write contracts for? As customers request machines, how

do I decide which machines to allocate to each customer? How many and which machines should be left idle or turned off? If I allocate this machine now, what is the risk that a higher-paying customer will arrive and there won't be any machines left?

We develop new algorithms that address all of these scheduling considerations. Furthermore, we show how to use our algorithms to compare different provisioning scenarios for a given workload and make good provisioning decisions.

Our results can be summarized as follows:

- It does matter which machines get used, especially when the data center is underutilized. Since most data centers provision enough capacity to handle their peak utilization, they have excess capacity at average utilization[18]. We present new algorithms that choose the right machines to maximize profit, both at high and low utilization of the data center. The new algorithms do even better when they try to anticipate the "riskiness" of their decisions, that is, the likelihood that more important customers will make requests soon.

- It also matters whether the unused machines can be turned off, rather than left idle. Idle machines consume roughly 50% of the power of machines at full utilization[14]. Simple heuristics like "turn off a machine that's been idle for 5 minutes" are sufficient to save a lot of money.

- We perform a sensitivity analysis of different kinds of customers and different types of machines. We draw some conclusions about which properties of a contract have the biggest impact on the ability of the data center to satisfy the customer. Namely, it is possible to satisfy some customers whose requests are very urgent, sometimes as many as three-quarters of all customers. It is even better if they will pay more for their urgent requests. Further, it is easy to make good choices even if the customers pay using as little as two different rates.

- Finally, having heterogeneity in the data center saves money compared to having only homogeneous machines. We show that the same set of customers can be satisfied at a lower cost and a higher profit in a heterogeneous data center using our new algorithms.

The rest of the paper is organized as follows. In the next section, we lay out our assumptions and our model of how the data center makes decisions. In Section 3, we describe our scheduling algorithms. Then we present our simulation framework and our results in Sections 4 and 5. In Section 6, we identify related work. Finally, we draw conclusions in Section 7.

2 Data center model

In this section we describe our model of a data center and lay out our assumptions about customers, data center providers, and the interactions between them. We also define our terminology.

2.1 Customers

Customers rent machines from the data center. The machines are used for one customer at a time and the customer gets the whole machine. If multiple virtual machines run on each physical machine, then machine here means each virtual machine.

Between customers, the machine must generally be reconfigured: disks wiped clean and new software installed. Whether the data center provider or the customer takes responsibility for setting up the machine, it takes time on the order of minutes or hours, not seconds. Therefore, customers rent machines for long blocks of time — a few hours to several days or a week. Their willingness to receive a machine after requesting one is similarly a few hours to days.

We use the word *request* to describe a customer asking for a single machine for a continuous block of time. The same customer may make many requests under the terms of the same contract. We use *request* instead of *job* or *task* to make it clear that one machine can only be used for one request at time. Task and job are often used for things like web page retrieval, where many of them from multiple customers can be run in parallel on the same machine. The scheduling of requests is otherwise similar to scheduling jobs onto processors.

2.2 Data center coordinator

The data center coordinator makes decisions for the data center. The basic problem that it tries to solve is to maximize the profit from a set of customer requests (a workload) and a set of machines. By comparing profits for different workloads, its solutions can be used to decide which workloads are best for a fixed set of machines. This desired workload can then be used to make decisions about which kinds of contracts to form with customers. By comparing profits for different sets of machines for a fixed workload, its solutions can be used to inform provisioning decisions.

Figure 1 shows our model of the data center. The data center coordinator has a *data center model* of the machines in the data center listing how many there are of each type. The *power model* tells it the number of watts each type of machine uses at different levels of CPU utilization, including when the machine is idle or off. (The power models we used for our experiments are shown in Table 2).

Customers arrive continuously and ask the data center coordinator for machines. For each customer request, the coordinator must first decide whether to accept or reject the

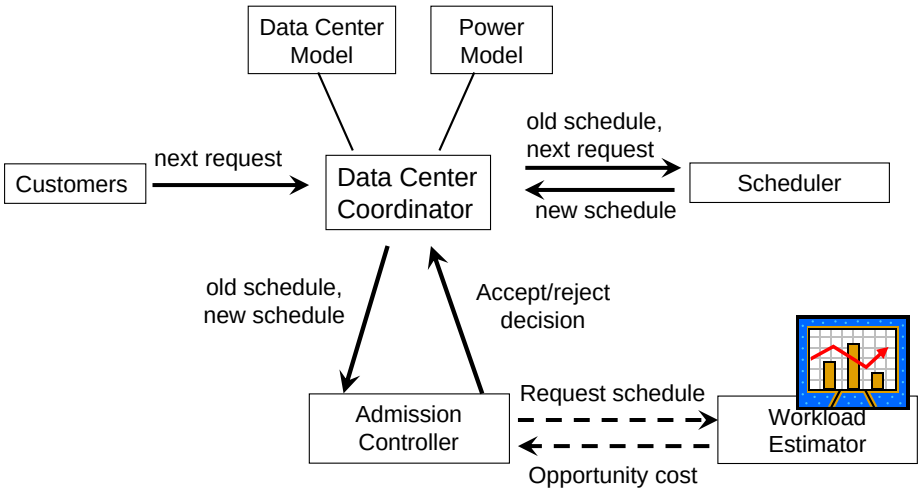


Figure 1. Data center coordinator. *Architecture showing communication internal to data center coordinator and outward to customers.*

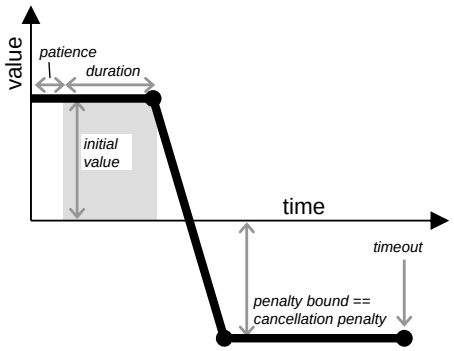


Figure 2. Utility function. *How much the customer is willing to pay for a request varies with when the request is scheduled.*

request. The *admission controller* makes this decision. Accepting (rejecting) the request is telling the customer that they will (not) receive a machine. If the coordinator accepts the request, it then allocates the machine to the customer (possibly at a later time, if the customer is willing to wait). The *scheduler* decides when to allocate the machine to the customer.

The coordinator does not know when customers will arrive or what requests they will make. It therefore must solve an *online* scheduling problem.

2.3 Customer requests

Each customer request consists of a utility function such as the one shown in Figure 2[2]. The utility function contains the desired length of time (*duration*) on a machine, the customer's *patience* to wait for a time slot, the price (value) that the customer is willing to pay, and the penalty that the customer will demand for non-fulfillment.

There may be a *rejection penalty* for rejecting the request, if the customer has a contract that requires the data center to accept a certain percentage of its requests. Or there may be a *cancellation penalty* for failing to schedule a request (actually allocate a machine within the correct time frame) after accepting it.

We believe that most requests will arrive in the context of a longer-term contract. The individual requests within the contract will share a utility function. Most customers only want to negotiate contract terms once or twice a year. Moreover, they may not want to think hard about what to pay, but simply choose a "gold standard" contract for urgent work with high guarantees at a high price or a "bronze standard" contract for less urgent work at lower price. We therefore use only a few different utility functions to model customers, although we do vary patience and price independently.

To avoid confusion, we call the price that the customer is willing to pay the *value* of the request. If the value does not exceed the data center's cost to fulfill the request, the coordinator will always reject the request. In earlier work[2] we discussed how general contract terms can be used to influence the value of each request. We then assume that the customer pays this price when the request is completed. There are other forms of price-setting, such as auctions, but they are outside the scope of this work.

2.4 Data center costs and profits

We are primarily concerned with data centers that contain heterogeneous machines. In this paper, we focus on energy costs. We model the energy cost for powering each machine, although we could double it to include the energy cost for cooling it. The *cost-rate* for a machine is the cost per watt per hour that we pay for electricity. Similarly, the *value-rate* of a request is the value per hour that the customer pays. Therefore, *profit-rate* = (*value-rate* − *cost-rate*).

2.5 Under vs. oversaturation

While the data center coordinator always tries to maximize its profit, the problem it tries to solve really has two cases: undersaturation vs. oversaturation of the data center. Both cases are interesting.

When the data center is undersaturated, the coordinator can accept and schedule all or close to all requests, so value is a constant. The key part of its decisions is choosing which machines will satisfy the requests on time at the lowest cost.

Many data centers are undersaturated because they are provisioned for peak usage while average usage is much lower[18]. In fact, part of the motivation for utility computing is to use the same data center to satisfy requests from different customers whose peaks rarely overlap. That is, the data center can allocate a machine to customer A during A's peak usage and then allocate the same machine to customer B during B's peak usage.

If the data center is over-saturated by requests, all machines are fully utilized and cost is a constant. The data center coordinator must prioritize requests from different customers to earn the maximum value, which generally means choosing the highest-valued requests. Trying to please too many customers results in broken SLAs and penalties. It is therefore better to make rejection decisions early, either before forming a contract or before accepting a request.

In the next section, we discuss the algorithms that the data center coordinator uses to make its admission and scheduling decisions.

3 Data center coordinator

Our data center coordinator makes its allocation decisions using the admission controller and request scheduler shown in Figure 1. Whenever a request arrives, the admission controller decides whether to admit or reject the request. The admission controller first invokes the scheduler to create a new schedule and then decides which is better between the schedules that include and omit the new request.

We first discuss different scheduling algorithms and then define what it means for a schedule to be "better" than another one. While the simple version of the admission controller simply takes the schedule with the higher profit, our new admission algorithm weighs the risk of rejecting future requests in its decisions.

3.1 Scheduling algorithms

We consider several different request scheduling algorithms.

fifo: The most basic scheduler is a standard first-in, first-out (FIFO) scheduler that assigns each new request to the next available machine of any type. This policy is oblivious to heterogeneity. If more than one machine is available at the same time, the next machine is chosen randomly from this group. *fifo* appends each new request to the existing schedule; it does not modify it. Therefore, any request that is already in the schedule will never be cancelled to accomodate later arriving requests.

fifo-profit: The *fifo-profit* scheduler is a heterogeneity-aware variant of *fifo*. Whenever there is a tie for the first available machine, this algorithm will choose the machine where the request will earn the highest profit. That machine is usually the cheapest machine available, unless it is unable to complete the request in time.

fifo-type: The *fifo-type* scheduler considers the first available machine of *each* type, rather than only looking at whichever types are available first. Like *fifo-profit*, it chooses the type where the request will earn the highest profit. Unlike *fifo* and *fifo-profit*, *fifo-type* is not work-conserving: some machines can sit idle while there are requests waiting to run on other (cheaper) machines.

fifo-opp: The *fifo-opp* scheduler is a *fifo-type* scheduler augmented to account for the opportunity cost of waiting for a machine of one type while other (more expensive) machines sit idle. Instead of choosing the machine with the highest profit, it compares machines by their (profit − opportunity cost). In this case, it uses the corresponding profit and opportunity cost for the block of time on each proposed machine. We describe how the opportunity cost is computed in the next section; it is the same opportunity cost that we calculate for admission control.

profit-rate: The *profit-rate* scheduler, inspired by [16], prioritizes requests by their profit-rates. Unlike the *fifo* variants, which never move a request that has been assigned to a machine, *profit-rate* will recompute the entire schedule whenever a new request arrives. *profit-rate* sorts all of the outstanding requests plus the new one by their profit-rate on the next available machine and chooses the request that earns the highest profit-rate. Note that requests earn different profit-rates on different types of machines, so they need to be kept in multiple sorted orders while computing the schedule.

best: The "*best*" scheduler is neither an online nor a realistic scheduler. Instead, it calculates an upper bound on the total profit achievable for a particular workload. *best* works offline: it receives information about all of the re-

quests in the workload at once, ignores time constraints (arrival times, deadlines, and penalties), and allows migration.

best sorts all of the requests in the workload in descending value-rate order and greedily packs the machines (like knapsacks with a capacity equal to the length of the simulation) in order from lowest cost to highest. If a machine can only partially fulfill a request, the remainder of the request is assigned to the next machine; this part is quite unrealistic because it splits the request onto two machines at very different times at no extra cost. Unused time on any machine is charged at the idle cost.

Any online or offline schedule constrained by arrival times and deadlines cannot earn more profit than that produced by *best*.

3.1.1 Discussion

None of the FIFO schedulers will move existing requests within a schedule as new requests arrive. These algorithms complete each request in the schedule at the time that was predicted when the request arrived, so they generally satisfy all requests when the data center is undersaturated. When the data center is oversaturated, however, so that not all requests can be satified, the FIFO algorithms select requests by their arrival time rather than by their profit.

Unlike the FIFO algorithms, the *profit-rate* scheduler may move a request around in the schedule many times before starting it, as better requests arrive in the interim. It is therefore likely to complete more of the higher-valued requests when the data center is oversaturated. However, a lower-valued request may get pushed so far back in the schedule that it misses its deadline. Cancellations are undesirable for several reasons, including maintaining good relationships with customers and the inclusion of cancellation penalties in SLAs.

All of the above algorithms may leave some machines unused some of the time, either because there are not enough incoming requests or because the values of the requests may not compensate the data center for the costs of running them. Most data centers will leave intermittently used machines idle.

We also consider a variant of each algorithm that turns the machine a off when it has been idle for one minute. When a machine is off, the algorithm assumes a minimum delay (we use five minutes) to reboot a machine before it can be used again.

3.2 Admission control algorithms

The goal of the admission controller is to limit the set of admitted requests to the set that will earn the highest profit. Hopefully, all admitted requests can be satisfied.

We explore two admission control algorithms. Both algorithms compare schedules that include and omit the request under consideration and choose one of them.

The *higher* admission control algorithm, as in [16, 2], chooses the schedule that earns more profit. However, the amount of additional profit is not considered, nor is the likelihood that the schedule will be too full to include future requests. This algorithm limits the number of requests admitted, but is unable to restrict the set of admitted requests to the most profitable ones.

Our second algorithm, *risk*, tries to predict the the arrival times and values of requests that will arrive in the near future so that it can weigh the *opportunity cost* of accepting this request. For each decision, the admission controller compares the potential increase in profit from accepting the request against the risk that more profitable requests will arrive but be rejected or delayed due to this request.

The pseudocode in Figure 3 shows how the opportunity cost is calculated for each request *R*. When a FIFO algorithm computes a new schedule, it appends to the old schedule. Wherever a request is first inserted is where it stays in the schedule. Therefore, the opportunity cost of a request is the profit we did *not* get because this request occupied a particular block of time on a particular machine.

The opportunity cost for each block of time on a machine is calculated using an estimate of how many higher-valued requests we expect to arrive between now and the end of that block, that would be scheduled onto a machine of the same type, and that would not be scheduled if this block of time were unavailable.

To predict future arrivals, the *workload estimator* keeps statistics about requests that arrived in the recent past. Each request is placed in a histogram bin determined by its value-rate. We use $\lceil \log_{1.05}(\textit{value-rate} * 100 + 1) \rceil$ to choose the bin and store a variable number of bins determined by the largest value-rate needed. For each bin, the histogram stores an estimate of the arrival rate and the average request duration.

We store a simple count of requests and a time interval to estimate the arrival rate, which is sufficient for workloads with a constant arrival rate. We could store an exponentially weighted estimate instead to adapt to changing arrival rates. The accuracy of the arrival rate estimate depends more on the number of requests that have arrived than on the interval duration. With the exponentially distributed arrival times in our workloads, about 30 requests appears to be sufficent to get a good estimate.

3.2.1 Example of calculating opportunity cost

We illustrate the algorithm with a simple example in a data center with 4 machines, all of type A. Each machine has a cost-rate of $\$0.50/\text{hour}$ for idle time and a cost-rate of $\$1/\text{hour}$ at full utilization. Suppose request *R* arrives at time *t* with a value-rate of $\$2/\text{hour}$, a duration of 4 hours, and a patience of 6 hours.

```
compute_opportunity_cost(request R)
{
    mtype = type of machine R scheduled on
    bin_R = R's value-rate bin
    lost_time = 0            // time not used for higher-valued requests
    opp_cost = 0             // opportunity cost so far
    Rlen = R.end - R.start
    for bin = bin_high downto bin_R + 1 {
        // req_rate is the arrival rate on R's machine from this bin
        req_rate = bin.arr_rate * percent_completed_on_type(mtype) /
                   num_machines_of_type(mtype)
        // compute number of requests that will arrive before R completes
        expected_requests = req_rate * (R.end - now)

        // if higher-valued requests take less time than R
        if (lost_time + expected_requests * bin.avgsize * timeFactor(mtype) < Rlen) {
            // add them to opportunity cost
            opp_cost += expected_requests * bin.avgValRate * bin.avgsize
            lost_time += expected_requests * bin.avgsize * timeFactor(mtype)
        }
        else {
            // else add only fraction of value prevented by R
            opp_cost += bin.avgValRate / timeFactor(mtype) * (Rlen - lost_time)
            lost_time = Rlen
            break
        }
    }
    // so far, opp_cost is just the value of the missed requests
    // subtract cost of fulfilling the missed requests and cost of idle time
    opp_cost -= lost_time * costPerMin(mtype)
    opp_cost -= (Rlen - lost_time) * idleCostPerMin(mtype)
    return opp_cost
}
```

Figure 3. The opportunity cost algorithm. *Calculating the opportunity cost of adding a request R to the schedule.*

The scheduler proposes the schedule depicted in Figure 4, in which R starts at time $t+4$ on M_0.

The workload estimator's current statistics are illustrated in Table 1. We need to estimate the total value we would miss because R is occupying M_0 from time $t+4$ to $t+8$.

We first look at the arrival rate for the requests of the highest value-rate, 4. There are 8 hours before R would complete, so we expect $8*0.1=0.8$ requests of value-rate 4 to arrive. However, because we have 4 machines, we assume only $1/4$ of those requests would go to M_0. Therefore, we expect $0.8*0.25*2=0.4$ hours worth of value-rate 4 requests to run on M_0 at a profit-rate of $\$4-1=\$3$. By running R, we would miss $\$3*0.4=\$1.20$ of profit from value-rate 4 jobs.

There are still $4-0.4=3.6$ hours left in R's time block, so we repeat the calculation for value-rate 3 jobs. That yields $8*0.25*0.25*2=1$ hour of value-rate 3 jobs for $(\$3-1)*1=\$2$ worth of missed profit on M_0 and $3.6-1=2.6$ hours left over. Since there are no other bins of higher value-rate than R, we subtract the idle cost for the 2.6 hours: $\$0.50*2.6=\$1.30$. The total opportunity cost of R is $\$1.20+2.00-1.30=\$1.90$.

Since R will yield a profit of $\$8-4=\$4$, the admission controller accepts R. Note that if R's scheduled start was further in the future or if we had fewer alternative machines, the opportunity cost for R would increase.

Now suppose M_0 and M_1 are type A machines but M_2 and M_3 are of type B. Rather than assume that requests are equally distributed among the 4 machines, we instead look at the percentage of the completed requests that were assigned to machines of each type.

Suppose that 80% of all requests are assigned to the cheaper type A machines. Then we expect half of the 80% to be assigned to each of M_0 and M_1. The ex-

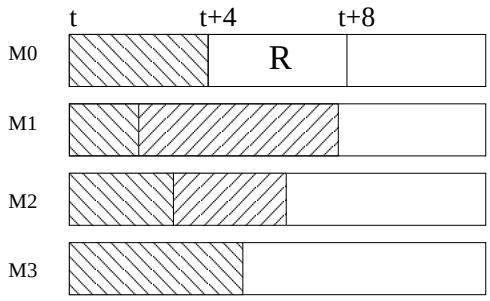


Figure 4. Risk example. *At time t we receive request R. The shaded regions represent requests that are already scheduled on machines M_0, M_1, M_2, and M_3.*

average value-rate	requests per hour	average duration
2	0.50	4
3	0.25	2
4	0.10	2

Table 1. Example workload estimates

pected duration of requests of value-rate 4 on M_0 is then $8*0.1*0.40*2=0.64$ hours at a profit of $\$3*0.64=\$1.92$. Similarly, the expected duration from requests of value-rate 3 is $8*0.25*0.40*2=1.6$ hours with a profit of $\$2*1.6=\$3.20$. The remaining idle time is $4-0.64-1.6=1.76$ hours with a cost of \$0.88. The total opportunity cost for R is then $\$1.92+3.20-0.88=\$4.24$ and the admission controller rejects R.

4 Simulation

We tested the algorithms using an event-driven simulator and a workload generator that we wrote. The workload generator creates requests according to distributions for value-rate, duration, patience and arrival-rate. The data center coordinator receives the stream of requests. For each request, it invokes the admission controller and scheduler to compute a new schedule and make an admission decision of accept or reject. If the request is accepted, the coordinator saves the new schedule. This new schedule is followed until the next accept decision is made (which supercedes it with a newer schedule). At the end of the simulation, we discard any unfinished requests when calculating the total profit and assume the machines remained idle or off instead of starting the requests.

We validated the simulator in two ways. First, we compared its results on a few small examples to those we derived by hand. Second, we used a completely separate code base for the offline algorithm *best* – it is simply an augmented knapsack solver. The simulator's results are quite similar to those from *best*, which gives us confidence in both sets of results.

machine class	100% util Watts	idle Watts	off Watts	time factor	power-perf
Power Model A					
BL	150	60	0	1.00	150
DL	200	100	25	1.25	250
BC	25	10	0	3.75	92
Power Model B					
B1	250	100	0	1.00	250
B2	200	100	0	1.25	250
B3	125	80	0	2.00	250
Power Model C					
C1	250	100	0	1.00	250
C2	133	65	0	1.50	200
C3	75	40	0	2.00	150

Table 2. Different power models

For our experiments, we ran simulations of 90 days. All times (request arrival, request duration, etc.) are at the granularity of 10 minutes and we set the price of electricity at 10 cents per kilowatt hour. We calculate the cost paid by the resource provider as simply the power cost incurred over the length of the simulation. We could double this number to (roughly) include the cost of power for cooling and add a fixed amount for the other components of total-cost-of-ownership, but we chose to focus only on the costs that our scheduling decisions can actually change. This focus makes the impact of our decisions clearer.

The parameters of the simulation fall into two categories: data center parameters and workload parameters. The data center parameters are the number of machines of each type and the power model for each type.

4.1 Power models

The power model includes points along the power-performance spectrum. For this paper, we focused on machines that are either fully (100%) utilized or idle, such as those used for multimedia rendering workloads and scientific computations. Our simulation can handle more points and we plan to include other power-performance points, such as those from using different power-states, also called dynamic voltage states (DVS), in the future.

Table 2 shows the power models we used. The first column names the type or class of machine. The next three columns list the number of Watts consumed by these machines at 100% CPU utilization, 0% CPU utilization (idle), and when turned off. Note that some machines consume power even when they are off; in some rack-mounted servers, the fan must continue to operate to dissipate heat from other nearby servers.

The next column, the *time factor*, shows the relative performance to expect on the machines. For example, in power model A, the BC-class machines have a time factor of 3.75, which means that the same tasks that take one hour to run on a BL-class machine will take 3.75 hours on a BC-class machine. Finally, the last column gives a sense for the relative power-performance merit of each machine; it is simply the Watts consumed at 100% utilization multiplied by the time factor.

Power model A represents three classes of machines that might be found in a data center today, one server and two kinds of blades. Most of our experiments use power model A. Power model B is used to illustrate what happens when machines of different speeds have the same power-performance ratio. Power model C represents three servers of different generations; the oldest generation has the lowest power-performance ratio.

4.2 Data center models

For each power model, we simulated a data center with 100 machines. In most experiments, one-third of the machines in the data center were of each type. In the final experiment, we also compared a data center containing only the newest, fastest machines to one where part of the budget was spent on new but slower and cheaper machines with a lower power-performance ratio.

4.3 Customer workload

The workload parameters create the utility functions for requests and also include the arrival rate of requests. In our simulation, the utility function is determined by the request duration, patience and value-rate, and whether there is a penalty for request cancellation and/or request rejection. Table 3 shows the parameter settings for the workloads used in our first experiment.

We chose these parameter setting to represent a mix of customers. DreamWorks, for example, has overnight workloads that typically last 13 hours where the number of machines required is sometimes known days or weeks ahead of time, and sometimes only a few hours ahead (especially if one data center experiences a failure and work must be shifted to another one)[21]. Closer analysis of DreamWorks' workload reveals that some machines are required for the entire 13 hours, but others are needed for less time, and the lengths of time can be predicted fairly accurately before the 13 hours commences. Finally, individual jobs within their workload have an average duration of only 2.36 hours. On some nights over the two months studied, the data center was fully saturated while on others (mostly weekends) the data center was quite undersaturated.

Another type of customer might be running an e-commerce site, such as hp.com. This customer needs a varying number of servers for its applications as its own workload demand changes. Complementary work to ours[20, 6, 18] decides how many servers are needed; increased demand is sometimes unexpected and urgent but often can be predicted days or weeks ahead of time based on diurnal, weekly, or seasonal patterns[18]. A customer running such a site might insist on rejection penalties, for example, during the December holiday shopping season when it sells much of its products.

parameter	value	frequency
interarrival time	0.1 hours	exponential
gold-high value-rate	4.4 cents/hour	25%
gold-low value-rate	3.2 cents/hour	25%
bronze-high value-rate	1.7 cents/hour	25%
bronze-low value-rate	1.2 cents/hour	25%
short duration	2.5 hours	50%
long duration	24 hours	50%
high patience	1 week	100%
medium patience	1.25 days	0%
low patience	8 hours	0%
superlow patience	2 hours	0%
penalty	cancellation	100%

Table 3. Workload parameter settings for base experiment

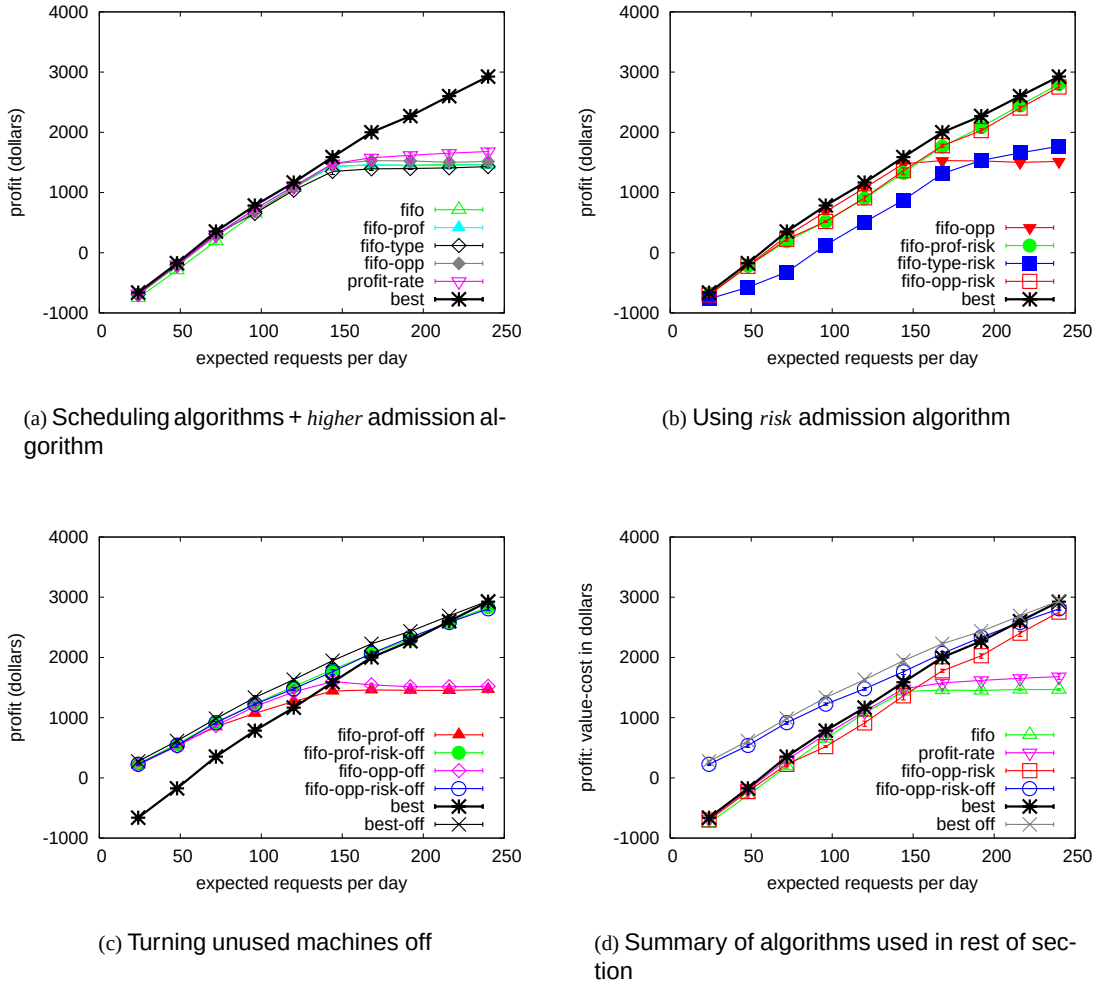
4.4 Assumptions

We make a few assumptions about scheduling requests to simplify the simulation. First, a request cannot be extended for a longer period once it has been scheduled. Second, once a request has started, it cannot be cancelled, suspended, or migrated. While virtual machines can be used for suspension and resumption or migration, virtual machines have other costs and are not yet integrated into our simulation.

5 Results

We ran many simulations to determine how the power model, data center model, and workload affect the ability of each algorithm to earn a profit. In this section we show results from some of the simulations. All graphs plot the average of five simulation runs.

We name the algorithms first by their scheduling algorithm, then by whether they used the *risk* admission control algorithm, then by whether they turned machines off or left them idle. For example, *fifo-profit* uses the FIFO-profit scheduling algorithm, the *higher* admission control algorithm, and leaves machines idle. *fifo-opp-risk-off* uses the FIFO-opp scheduling algorithm, the *risk* admission algorithm, and turns machines off.



(a) Scheduling algorithms + *higher* admission algorithm

(b) Using *risk* admission algorithm

(c) Turning unused machines off

(d) Summary of algorithms used in rest of section

Figure 5. Basic experiment. *Exploring the effects of different scheduling and admission control algorithms with a fixed workload, data center, and power model.*

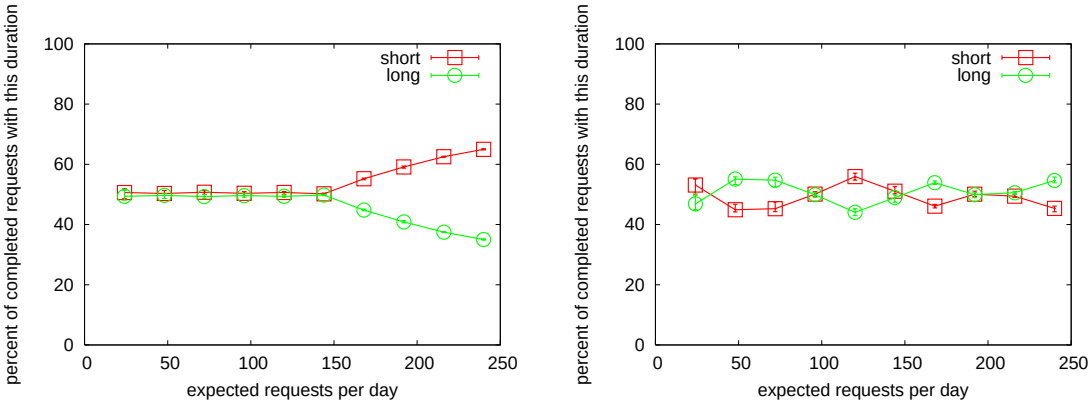
5.1 Basic experiment

We first study the effect the load on the data center has on the profit obtained by each algorithm. All other workload parameters are fixed at the values in Table 3 and we simulate a data center with 100 machines that conform to power model A.

Heterogeneity-aware scheduling: Figure 5 shows the results for all of our algorithms. In Figure 5(a), we compare the different scheduling algorithms, all of which used the *higher* admission control algorithm. They all perform quite well, very close to *best*, until the data center reaches saturation. At about 150 requests per day, the data center no longer has the capacity to satisfy all requests. While the algorithms vary slightly from each other in their ability to prioritize the higher-valued requests of those accepted, the *higher* admission control algorithm rejects many of the highest-valued requests: it has already filled the schedule with lower-valued requests that the *fifo-** variants will not cancel and that profit-rate pays so much to cancel that it barely compensates their higher-value. These algorithms therefore have a flat profit curve above saturation, because they essentially admit and schedule requests based on their arrival times rather than their values.

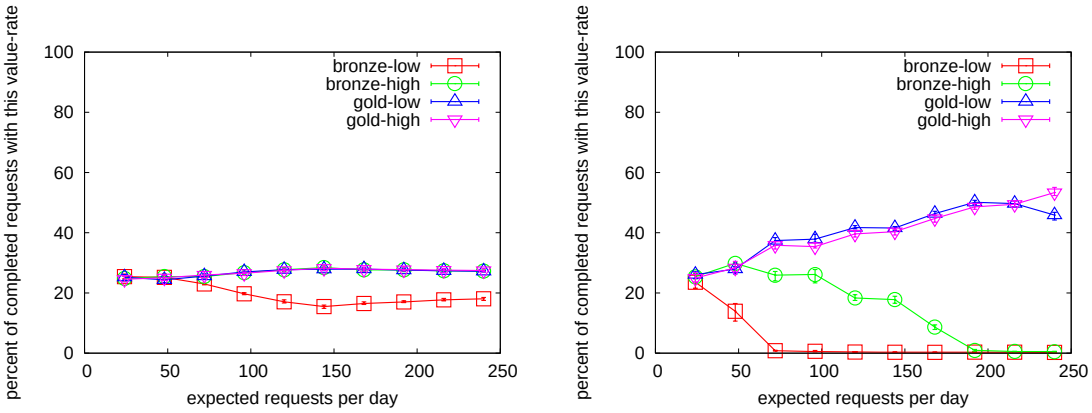
***Risk* admission control:** The solution lies in better admission control, as shown in Figure 5(b), which repeats *fifo-opp* for comparison and then shows the other scheduling algorithms with *risk* admission control. (There is no *profit-rate-risk* combination because calculating the opportunity cost is much harder when a request can move its position in the schedule.)

We first notice that *fifo-profit-risk* and *fifo-opp-risk* now do quite well above saturation; both algorithms are very



(a) fifo-prof-off (b) fifo-opp-risk-off

Figure 6. Fate of long vs. short duration requests. *fifo-profit-off prefers short duration requests, while fifo-opp-risk-off does not discriminate by duration.*



(a) fifo-prof-off (b) fifo-opp-risk-off

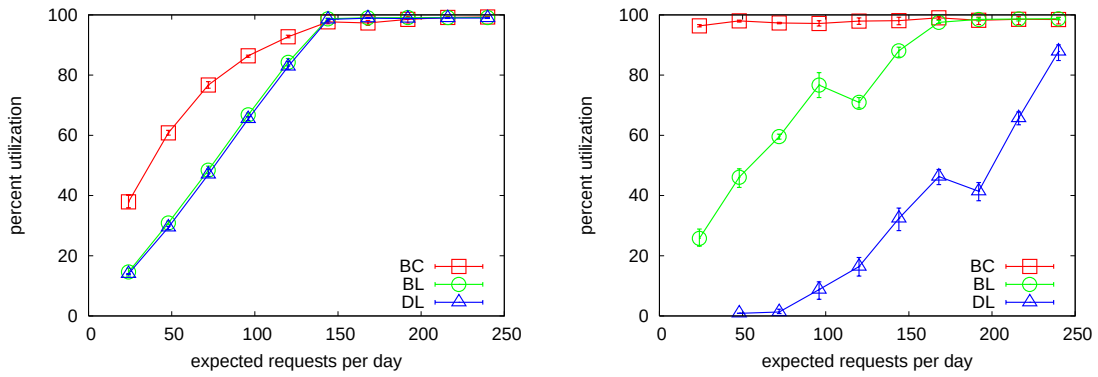
Figure 7. Fate of requests by value-rate. *fifo-opp-risk-off is able to accept more requests of the highest value-rates.*

close to *best*. They are similar to each other because the opportunity cost of leaving a machine of one type idle now and filling another, cheaper, machine's schedule a long time out rarely makes it worth while to wait for a cheaper machine. Therefore, *fifo-profit-risk*, which never waits, and *fifo-opp-risk*, which weighs the opportunity cost, make similar decisions. *fifo-type-risk* does not do as well because it *will* wait for a cheaper machine without considering the opportunity cost.

Turning machines off: In Figure 5(c) we show the algorithms that turn machines off when they have been idle for ten minutes. When the data center is filled to capacity, machines are essentially never idle and so there is no impact on profit. However, when the data center is undersaturated, machines are idle. Turning machines off then makes the difference between an operating profit and a loss: note that the y axis is below zero at low arrival rates unless machines are turned off. (Whether the data center should have been provisioned differently is a different question; it may be that it is only undersaturated some of the time.)

Algorithms to compare: Figure 5(d) repeats the algorithms that we will compare for the rest of this section. *fifo-profit-risk-off* is our best online algorithm for this workload. *profit-rate* is the algorithm advocated in prior work[2] and *fifo* is the naive, worst-case algorithm that is unaware of heterogeneity. While *best-off* is not an online algorithm nor does it produce a followable schedule, it does provide an upper bound on profit. The profit from *fifo-opp-risk-off* is within 10% of *best-off* is nearly all cases.

Durations of requests: The next set of graphs illustrate



(a) fifo-prof-off (b) fifo-opp-risk-off

Figure 8. Utilization of machines of different types. *fifo-opp-risk-off prefers the less expensive machines and accepts fewer requests because it only schedules requests on machines on which they are profitable.*

why *fifo-opp-risk-off* performs so well. Since we want to highlight the contribution of our opportunity cost calculation, we compare it to *fifo-profit-off*, which does not have one. Figure 6 shows the fate of requests by duration. *fifo-profit-off* favors short requests: a short request is more likely to fit onto a slower, cheaper machine. *fifo-opp-risk-off* accepts requests of different durations in proportion to their appearance in the workload (which is half long, half short), which leaves it better able to discriminate among requests based on their value.

Value-rates of requests: The fate of requests by their value-rate is shown in Figure 7. While both *fifo-profit-off* and *fifo-opp-risk-off* are able to discriminate against the lowest valued requests, which are only profitable on the slowest, cheapest machines, *fifo-opp-risk-off* is able to strongly prefer the requests of the highest values. *fifo-opp-risk-off* does so well because the opportunity cost calculation anticipates the arrival of these highest valued requests and essentially saves space in the schedule for them.

Utilization of different machine types: Finally, Figure 8 compares the utilization of machines of different types by *fifo*, the algorithm which is oblivious to machine types, and *fifo-opp-risk-off*, which combines all three of our optimizations. First, by being heterogeneity-aware, *fifo-opp-risk-off* is able to prefer the lowest cost BC-class machines: those machines are saturated first while the highest cost DL-class machines are saturated last. *fifo*, on the other hand, uses BC-class machines only slightly more than the other machines, and mostly because requests stay on the BC-class machines for a much longer period of time. Second, *fifo-opp-risk-off* requires a much higher arrival rate to saturate the data center. Only the gold-high and gold-low value-rate requests can earn a profit on the DL-class machines.

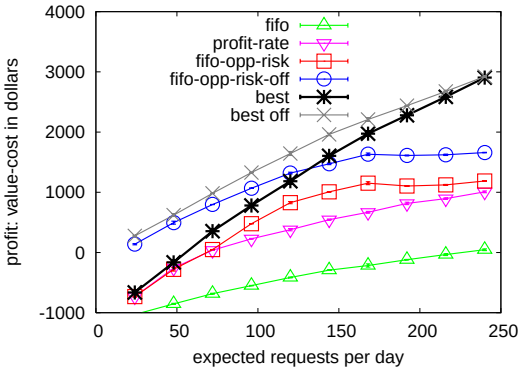


Figure 9. All customers want machines urgently:. *75% have superlow patience (within 2 hours); 25% have low patience (today).*

Since the *risk* admission control algorithm essentially reserves space for the highest-value requests on the lowest-cost machines, there must be enough high-value requests to saturate all machines before the DL-class machines are used. Therefore, *fifo-opp-risk-off* earns more profit than the other algorithms while also using fewer machines.

5.2 Urgent customers

We now turn to experiments that vary the workload parameters, one at a time. While the customers in Figure 5 are patient, the customers in Figure 9 are not. *best* and *best-off* are oblivious to deadlines, so their performance is unaltered. The other algorithms perform worse when a new request cannot wait for a few other requests to be scheduled ahead of it: even though *fifo-opp-risk-off* predicts the average ar-

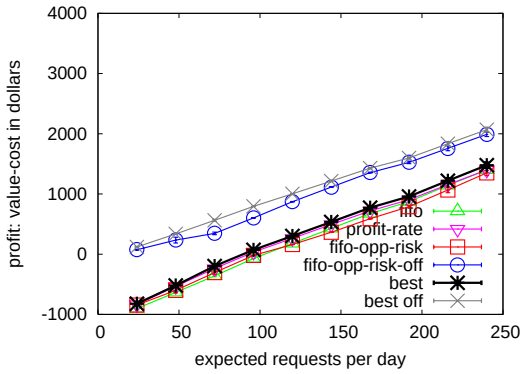


Figure 10. Many customers want to pay a low rate:. *75% bronze low value-rate, 25% gold high value-rate.*

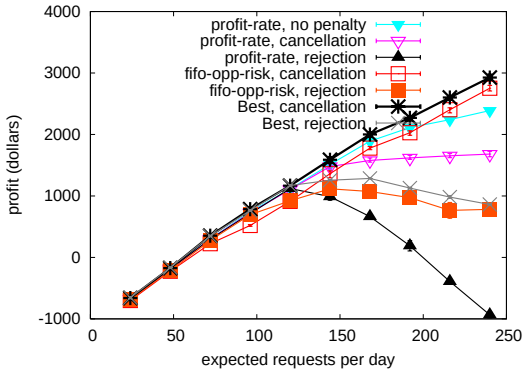


Figure 11. Impact of penalties:. *compare profit-rate and fifo-opp-risk with rejection, cancellation, and no penalties.*

rival rate correctly, the actual arrival times are exponentially distributed and requests do arrive in bursts. The algorithms cannot satisfy a burst of customers who all want machines at once. Better prediction methods might leave more room in the schedule.

Although a high percentage of customers who will not wait for *any* other requests to be scheduled ahead of them cause performance degradation for our algorithms, we did not see any difference in performance when we tried a mix of superlow and medium patience customers. Similarly, we did not see any degradation when customers exhibited any mix of low, medium, and high patience requests, including 100% low patience requests.

5.3 Cost conscious customers

In Figure 10, we consider what happens when most customers want to pay the lowest value-rate, which is only enough to make a profit on the slowest, cheapest machines – in this experiment, only on the BC-class machines. First, with so many lower value requests, it is harder to saturate the data center; many requests cannot be run profitably on the highest cost machines and are rejected. Second, even when all requests can be scheduled, the profit is lower since the total value of what is scheduled is lower. Finally, low-value requests can only be satisfied when the data center has excess capacity of the cheapest machines.

5.4 Impact of penalties

Different contracts may impose different penalties for rejecting or cancelling requests. In Figure 11 we examine the impact of these penalties. First, all of the algorithms, even *best*, suffer tremendously from rejection penalties when the data center is oversaturated. If the customers can expect rejection penalties, then the data center provider should limit the number of contracts so that there is no oversaturation, presumably by requiring the customers to pay

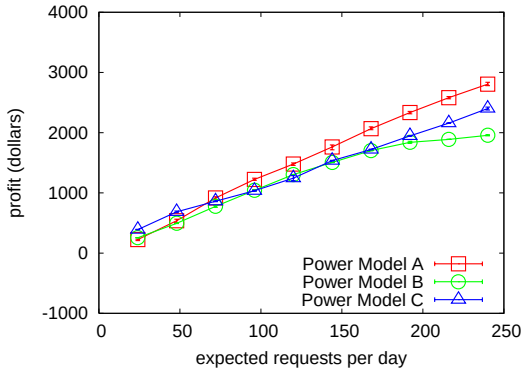


Figure 12. Different power models:. *compare the profit available from different power models, all using* fifo-opp-risk-off.

more. AuYoung[2] considers different contract admission policies, where contract admission precedes request admission.

Second, *fifo-opp-risk* never pays any cancellation penalties; if it accepts a request, it is because the request fit into the schedule. Since new requests only get appended to the schedule, no requests are ever cancelled.

Third, when *profit-rate* does *not* pay any penalties, it performs quite well, comparably to *fifo-opp-risk* except under extreme oversaturation. This result both highlights the large number of requests that *profit-rate* cancels and suggests that the very simple *profit-rate* algorithm is sufficient for customers who are indifferent about whether their requests are granted.

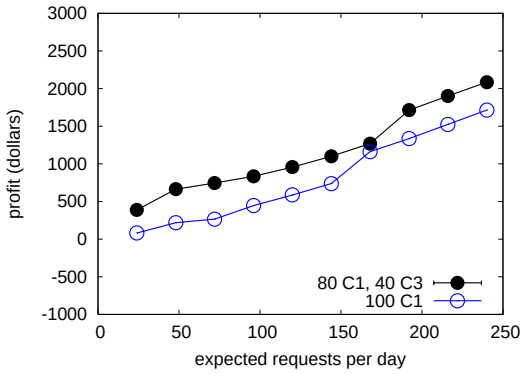


Figure 13. Provisioning choices:. *compare two different ways of provisioning the same data center.*

5.5 Different power models

The next two experiments compare different data centers running the workload from Section 5.1. In this experiment we compare data centers whose machines conform to the three different power models in Table 2. Each data center has equal numbers of the three machines in its power model. (We raised the values of the requests for power models B and C proportionately so that each workload was capable of earning the same profit.) Figure 12 shows the results. We see that power model B, whose machines all have the same power-performance metric, has the least opportunity to save money when the data center is undersaturated and the least ability to reserve the fastest machines for the highest-value requests. Power model A, which has the greatest difference between the highest and lowest power-performance metrics in the data center, is able to earn the most profit.

5.6 Provisioning choices

In this last experiment we return to provisioning choices. Power model C represents servers of three different generations. We now compare replacing all of the machines in the data center with 100 new C1 machines to getting 80 new C1 machines but continuing to use 40 older C3 machines. (We chose 40 C3 machines so that the data center would have the same capacity either way; the C1 machines are exactly twice as fast as the C3 machines.) The results in Figure 13 show that by using the older machines, which have a much lower power cost, it is possible to save a lot of money. For example, when the arrival rate is 200 requests per day, the data center is operating at full capacity and using the older C3 machines increases profit by about 30%.

6 Related work

There is much prior art on scheduling algorithms; we only attempt to cover work on very similar problems. This work falls into the three categories below.

Profit-maximizing online scheduling algorithms: Millenium[7] and RiskReward[10] use utility functions to performs an online assigment of jobs (requests) to machines. They try to maximize the value of the jobs completed, but do not consider their cost. Popovici[16] and AuYoung[2] add cost functions proportional to machine usage on homogeneous machines and try to maximize profit. They assume that machines are available for intermittent usage and do not pay for time when they are not used or for time to reboot or to acquire a new machine.

AuYoung[2] additionally looks at jobs in the context of contracts, which can impose constraints like "you need to accept all requests from this customer." While we confirm their conclusion that *profit-rate* is better than *fifo*, they only explore the *higher* admission control algorithm.

Scheduling algorithms that predict risk of schedule: Although RiskReward[10] has a notion of the opportunity cost of a schedule, they define it in terms of prioritizing two known requests by the relative decay in their utility functions. They do not consider the effects of future requests on the current schedule; in fact, they do not consider admission control separately from scheduling.

Iyer and Druschel[11] use short pauses in their schedules to wait for future work, in their case, future disk read and write requests. The length of the pause is determined by arrival rate statistics similar to ours, and taking the pause is similar to our leaving space in the schedule for future requests, although there is no notion of value-rate for their requests.

The theoretical problem most similar to ours is the online knapsack problem where each admittance decision is based on a moving threshold for the item's size[12]. Setting the threshold is similar to predicting the opportunity cost for a given request, although it does not have to worry about time constraints, multiple machines (knapsacks), or heterogeneous costs.

Scheduling to conserve power usage: Finally, there has been much work lately on conserving power usage. Rajamani[17] provides a good overview of using different power-states to conserve power. Other work has focused on turning (mostly homogeneous) machines off to save power[5, 15, 13].

Heath[8, 9] considers turning off heterogeneous machines and also present algorithms to decide how many machines are needed for a given application load. Both Chen[6] and Rusu[19] consider both using power-states and turning machines off. However, Heath, Rusu, and Chen use a different model of customer requests than we do. Rusu's requests are short and overlapping, Chen makes allocation decisions based on a response time SLA, and Heath is simply required to run a particular workload. We consider long-term allocations of machines and we are not required to accept all requests; we therefore focus on improving profit

rather than simply reducing costs.

7 Conclusions

As data centers evolve to serve more diverse customers sharing more heterogeneous hardware, it becomes increasingly important to design scheduling algorithms that match workloads to resources. In this paper, we propose several new algorithms for a cost-aware provider to maximize its profit as it makes admission and scheduling decisions for customer requests. These algorithms incorporate heterogeneity-awareness and risk and we evaluate them by simulating a variety of customer workloads on data centers of different sizes with different combinations of machines.

Our results demonstrate several interesting insights. First, we show that awareness of heterogeneity can improve profit, particularly when the datacenter is underused. Second, we show that an awareness of the opportunity cost of scheduling decisions improves profit when the data center is saturated. Third, we demonstrate that simple heuristics to turn machines off save even more money. In most cases, in addition to providing significant increases in net profit, our algorithms perform fairly close to an "unachievable best" algorithm that we use for comparison. Finally, we also study the effects of varying the workload and data center parameters (customer patience, value-rates, penalties, and power models) and show that our algorithms can inform provisioning decisions for the data center.

We would like to extend our algorithms in several directions. First, we would like to support additional voltage and frequency states. Second, we would like to allow additional scheduler constraints, such as allocating the same machines to the same customer repeatedly for security or performance reasons. Third, we would like to include other components of the total cost of ownership, such as cooling, real-estate, personnel, and software licenses. Finally, in this paper, we consider only one data center provider. Future extensions may consider multiple data centers competing in a market-based model.

References

[1] Advanced configuration and power interface.

[2] A. AuYoung, L. Grit, J. Wiener, and J. Wilkes. Service contracts and aggregate utility functions. In *High Performance Distributed Computing (HPDC)*, June 2006.

[3] L. Barroso, J. Dean, and U. Holzle. Web search for a planet: the google cluster architecture. *IEEE Micro*, 23(2), 2003.

[4] C. E. Bash, C. D. Patel, and R. K. Sharma. Dynamic thermal management of air cooled data centers. In *Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2006.

[5] J. S. Chase, D. C. Anderson, and A. M. V. P. N. Thakar. Managing energy and server resources in hosting centers. In *Symposium on Operating Systems Principles (SOSP)*, 1999.

[6] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *SIGMETRICS*, 2005.

[7] B. N. Chun and D. E. C. (UCB.). User-centric performance analysis of market-based cluster batch schedulers. In *Symposium on Cluster Computing and the Grid*, May 2002.

[8] T. Heath, B. Diniz, E. V. Carrera, and W. Meira Jr. Self-configuring heterogeneous server clusters. In *Workshop on Compilers and Operating Systems for Low Power (COLP)*, 2003.

[9] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2005.

[10] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *Symposium on High Performance Distributed Computing (HPDC)*, 2004.

[11] S. Iyer and P. Druschel. Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O. In *Symposium on Operating Systems Principles (SOSP)*, 2001.

[12] G. S. Leuker. Average-case analysis of off-line and on-line knapsack problems. *Mathematical Programming*, 29(2), 1998.

[13] L. Mastroleon, N. Bambos, C. Kozyrakis, and D. Economou. Autonomic power management schemes for internet servers and data centers. In *IEEE Global Telecommunications Conference (GLOBECOMM)*, 2005.

[14] J. Moore, R. Sharma, R. Shih, J. Chase, C. Patel, and P. Ranganathan. Going beyond CPUs: the potential of temperature-aware data center architectures. In *Workshop on Temperature-aware Computer Systems*, 2004.

[15] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on Compilers and Operating Systems for Low Power (COLP)*, 2001.

[16] F. I. Popovici and J. Wilkes. Profitable services in an uncertain world. In *Supercomputing*, November 2005.

[17] K. Rajamani and C. Lefurgy. On evaluationg request-distribution schemes for saving energy in server clusters. In *IEEE Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2003.

[18] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. QoS-driven server migration for internet data centers. In *Real-time Systems Symposium (RTSS)*, 2002.

[19] C. Rusu, A. Rerreira, C. Scordino, A. Watson, R. Melhem, and D. Mossé. Energy-efficient real-time heterogeneous server clusters. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2006.

[20] Z. Wang, X. Zhu, and S. Singhal. Utilization and SLO-based control for dynamic resizing of resource partitions. In *Distributed Systems: Operations and Management (DSOM)*, 2005.

[21] Y. Zhou, T. Kelly, J. Wiener, and E. Anderson. An extended evaluation of two-phase scheduling methods for animation rendering. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2005.