QPRED: Using Quantile Predictions to Improve Power Usage for Private Clouds

Rich Wolski
Computer Science Department
University of California, Santa Barbara
Santa Barbara, CA, USA
Email: rich@cs.ucsb.edu

John Brevik
Department of Mathematics
California State University at Long Beach
Long Beach, CA, USA
Email: John.Brevik@csulb.edu

***Abstract*—In this paper we describe a new, efficient predictive scheduling methodology for implementing computing infrastructure power savings using private clouds. Our approach, termed "QPRED," estimates the quantiles on the distribution of future machine usage so that unneeded machines may be powered down to save power. A cloud administrator sets a bound on the probability that all available machines will be powered down when a cloud request arrives. This target probability is the basis of a Service Level Agreement between the cloud administrator and all cloud users covering start-up delay resulting from power savings. Our results, validated using activity traces from several private clouds used in commercial production, indicate that QPRED successfully reduces power consumption substantially while maintaining the SLAs specified by the cloud administrator.**

***Keywords*-cloud power optimization, cloud workload, performance evaluation**

I. Introduction

Cloud computing, in the form of "Infrastructure as a Service" (IaaS), has emerged as a new methodology for organizations to manage digital assets and the physical computing infrastructure that hosts them. Public clouds, such as Amazon's AWS [1] and Google Cloud Platform [2], rent virtual machines (VMs), network connectivity, and storage via web services APIs over the Internet. Customers of public clouds use an e-commerce-style interface to obtain these rentals in a way that is fully automated and self-service.

On the other hand, private clouds built using technologies such as Eucalyptus [3], [4], OpenStack [5], and CloudStack [6] operate in private datacenters, each under the control of an organization's Information Technology (IT) staff. They offer the same automated self-service interfaces as public clouds but to employees under a quota-controlled charge-back accounting system rather than to billed customers. Thus private clouds are a way of using e-commerce technologies to automate and streamline IT management of private datacenters through e-commerce-style self-service.

In this paper, we describe a scheduling methodology that is designed to save electrical power in cloud settings using on-line, non-parametric predictions of future demand. Cloud operators must be able to offer Service Level Agreements (SLAs) to their users so that these users can reason about how applications will behave when they are hosted, just as they do when these applications are run on physical infrastructure. Our approach allows the cloud administrator to make a probabilistic guarantee regarding the impact that power saving will have on user experience.

Clouds, by their very nature, obscure the specific infrastructure characteristics from the infrastructure users in the form of abstractions. Users reason about cloud use in terms of SLAs associated with its abstractions and experience the cloud in terms of delivered performance. Powering off idle servers carries with it the potential for a user-perceived delay during virtual machine (VM) start-up if a physical server needs to be powered on before the VM can start. Deskside- and laptop-class hardware can "hibernate," thereby minimizing this delay, but full hibernation support is not available for many commercial-grade servers. As a result, unused machines must be fully powered down to save power. With large memories and disk subsystems, the power-up delay associated server class machines can be significant (tens of minutes in some cases). Thus our methodology attempts to save as much power as possible subject to a probabilistic SLA that the cloud administrator sets with respect to the additional power-up delay a user might experience which is advertised to her or his users.

The key to our approach is the ability to make a conservative prediction of an arbitrary quantile from the distribution of machines that will be needed a short time into the future. Similar to [7], [8], we use a new fast, non-parametric prediction algorithm to estimate quantile bounds from measurement samples over fixed time epochs. Our methodology monitors cloud activity and uses the quantile prediction to estimate how many "hot spares" will be needed to host VMs that will be requested in the next time epoch. All other machines in the cloud not in use are then powered down.

The methodology is novel in that it does not rely on the *a priori* assumption that the "random" quantities of interest obey well-behaved and simple statistical distributions (*e.g.*, that process lifetimes are exponentially distributed). Comparable approaches [9], [10] employ sophisticated statistical models for queue wait times, workload, *etc.* based on distri-

This work was supported, in part, by NSF grants ACI-1541215, CNS-0905237, CNS-1218808, and ACI-0751315.

butional assumptions that enable computational tractability.

When a request to start a VM is initiated, the methodology first looks for a machine that is already powered up to host the VM. If no such machine is available, the scheduler will delay the VM request pending the power up of a dormant machine and the power-up time is experienced by the user as additional start-up delay. The quantile estimate allows the cloud administrator to set the maximum probability that no machine will be powered up and ready when a user request for service is initiated. The result is that the user experience is perturbed by a predictable fraction of the total request population. That is, the probability of finding no machine powered-up and available (defined by the quantile the administrator chooses) determines the maximum fraction of total user requests that will experience some form of delay. We term these quantile-predicting schedulers *QPRED* schedulers.

We validate the overall QPRED scheduling methodology using Eucalyptus [3], [4], an open-source platform for implementing private clouds in production datacenters and with various product workloads gathered from commercial "big data" hosting companies.

Eucalyptus is used commercially, and several of its commercial customers have made traces of their respective workloads available (under the condition of anonymity) for the purposes of evaluating our approach [1]. Thus the results we present herein depict effects that are observed from "real-world" production private cloud settings. In addition to understanding the degree to which power management could benefit this category of cloud usage, we are also interested in an algorithm that can be made to work "on-line" as part of the resource scheduling implementation. QPRED uses a short history of single-valued measurements (typically no more than 1000) that it must consult in sorted order and an incrementally updated running calculation of the average cloud request interarrival time. Thus its implementation can be made highly efficient with respect to computational and required memory state.

Our results indicate that the QPRED methodology can provide the cloud administrator with the abilty to offer a start-up SLA while also resulting in substantial power savings. Thus, while the problem of power management in data centers has been extensively studied [11], [9], [12], [13], [14] our work is the first to detail the efficacy of an efficient on-line statistical prediction strategy that provides a start-up SLA using production commercial private cloud workloads. It is unique in its use of an algorithm that can be implemented with minimal computational and storage requirements.

II. VM Scheduling and Power Consumption

Because clouds must be able to manage workloads scalably, their scheduling algorithms must be efficient. Eucalyptus, for example, only includes schedulers (Greedy and Round-robin) that assign a VM to a node at the time the request for the VM arrives at the cloud from the user. Further, because VM migration can require substantial intra-cloud bandwidth, each scheduler makes only a single placement decision for a VM at the beginning of a VM's lifetime.

Using only the ability to power machines on and off, the problem of optimizing power usage in this scheduling scenario without denying access (*i.e.*, turning away VM requests when machines are available but powered down) can be solved trivially: All machines are powered down until they are needed to run a VM. When a VM request arrives at the scheduler, the scheduler attempts to assign it to a node [2] that is already powered up. If no node is located, the scheduler chooses a node that is powered down, sends it a power-up signal, and launches the VM on the node once it has been successfully powered on. If the scheduler uses a "greedy" strategy – one that "fills" nodes with incoming VM requests before selecting a new node – this methodology is optimal with respect to power consumption under the constraints that

- each VM is considered once
- the scheduler makes only one placement decision for each VM at the time the VM start request arrives, and
- no additional information beyond what is needed to determine the capacity required for each VM is provided.

The scheduling complexity of such schedulers is $O(n * m)$ for n VMs and m machines (each of m machines might need to be considered for each of n VMs worst case). Because $n >> m$ in most cloud settings, we consider this to be $O(n)$ complexity.

Because this strategy waits to power up nodes until they are needed, VMs that cannot be started until a node has been fully powered on must also wait; this added delay is experienced by users until their VMs become available for use. Machine power-up times, particularly for server-class machines, can be lengthy: Depending on the machine's configuration, it may require as much as 30 minutes to go from a powered-off state to one in which a VM can be started. Moreover, Eucalyptus makes heavy use of caching and copy-on-write techniques to reduce VM launch times. A cached 10-gigabyte VM can be launched in under a minute if the local disks are server class. Thus a scheduling strategy that tries to optimize power usage may also introduce VM start-up overhead that is dramatic and may be unacceptable for some applications or users.

We formulate the problem of moderating power consumption in terms of a tradeoff between the probability that a VM (and its user) will experience a start-up delay and the power saved by having machines powered down.

[1] The traces are available from http://www.cs.ucsb.edu/~rich/workload/

[2] We will use the terms "machine" and "node" interchangeably to refer to a machine configured into a cloud that is running a hypervisor and can host a VM that is started and terminated by a user making requests to the cloud.

Well-written cloud applications are typically prepared for variation in VM start-up delay as long as the delays occur relatively infrequently. Thus our approach is to allow the cloud administrator to set a maximum target probability for any given VM to experience a start-up delay because a machine needs to be powered up. The scheduler must then keep enough "extra" machines ("hot spares") powered on so that the probability that a VM start request will arrive while no powered-on machines are available is at or below the target. At the same time, the scheduler must maintain $O(n)$ complexity to avoid introducing unacceptable or unpredictable overhead if the load scales.

Notice that this formulation of the scheduling problem prioritizes user experience in the form of minimized VM start-up delay over power savings. That is, we investigate schedulers that are designed to implement a Service Level Agreement (SLA) in terms of VM start-up times between the cloud's users and the cloud's operators while at the same time minimizing power usage subject to the SLA. This user-centric approach based on SLAs is typical for private cloud deployments.

Notice also that simply keeping a single or a fixed number of hot spares may not provide enough additional powered-on capacity if VM arrivals fill and exceed the capacity of the spares before a new spare can be fully powered on. As an example, suppose that the scheduler attempts to keep a single hot spare available, that each node in a cloud can host 8 single-core VMs, and that the machine power-up delay is 600 seconds. If 16 single-core requests arrive in a 600-second interval and there is only one hot spare, at least one VM will experience a start-up delay. Further, the cloud administrator cannot predict nor control the rate at which VMs (and users) experience start-up delay with this approach, making it difficult to provide a reliable SLA.

Thus, we investigate $O(n)$ scheduling methods that make a *prediction* of the number of additional machines that must be powered on at any moment so that the maximum target probability specified by the cloud administrator (*i.e.*, the SLA) for VM start-up delay will not be exceeded.

The Prediction Method

The goal is to use the information provided by the history of node occupancy to predict the number of nodes that will be required going forward and therefore the number of hot spares to keep on hand. To this end, we poll the system at regular intervals. To be sure, over a particular time interval there will likely be points in time when there are fewer or more nodes occupied; the number with which we will be concerned is the *maximum* number of nodes simultaneously occupied during the time interval.

In principle, the time-series information used for this inference will consist of an $(N+1)$-by-$(N+1)$ matrix of transition probabilities (for a cloud with N nodes) between all possible numbers, including 0, of occupied nodes. This formulation defines a large number of transition probabilities, even for a modest-sized cloud, to estimate from a sample of any reasonable size. (On the other hand, the majority of the transitions are vanishingly improbable, as the transitions themselves will tend to be small, provided that the sampling interval is short enough, so the number of useful probabilities to be estimated, while still substantial, is not extremely large.)

In the data sets we have studied, the transition probabilities are almost completely captured by the probabilities of *differences* from one interval to the next. That is, for example, given that there are 4 occupied nodes in one interval, the probability of going to 6 at the next interval is very nearly identical to the probability of going from 1 to 3 or 5 to 7. This behavior allows us to use a much simpler time series, namely that of the difference in the number of occupied nodes from one time step to the next. This simplification, in fact, removes the time-series character from the problem entirely: If we would like to be, say, 95% certain that we will have enough hot spares to handle the incoming jobs for the next time interval, we need only look at some estimate (in the statistical sense) for the 0.95 quantile of the set of differences. (In this work, for quantile inference, we simply use the percentile from the current measurement history although it is possible to use confidence upper bound, easily calculated from the order statistic via binomial means [7], [8], as a conservative estimate if necessary.) As a simple example, if we have inferred that this quantile is less than $+2$, this reflects the belief that there is at least a 95% probability that the number of nodes required at the next time step will be no more than two greater than the number needed in the current time step. Thus, keeping two hot spares on hand will supply us with the desired confidence of having enough resources ready to handle incoming work without delay.

We implement the QPRED prediction methodology using a doubly linked list and a red-black tree, each holding the maximum difference in busy machines recorded over an epoch. Figure 1 depicts these data structures graphically. At the end of each epoch, the latest (youngest) difference of maxima (henceforth called simply the "difference") is added to one end of the linked list and the oldest difference is removed. Similarly, the youngest difference is added to the red-black tree (so that differences are kept sorted) and the one that is removed is also deleted from the tree.

The history size (number of entries) is a fixed parameter supplied during configuration of a predictor. The total *time* covered by the history is the product of the number of entries in the history size and the epoch length.

To compute a prediction of the q^{th} quantile of the differences with a history size of H, the methodology extracts the entry corresponding to the $(1-q)\cdot H$ largest value in the red-black tree. For example, if $H = 100$, and $q = 0.95$, then the 5^{th} largest value in the red-black tree is the prediction

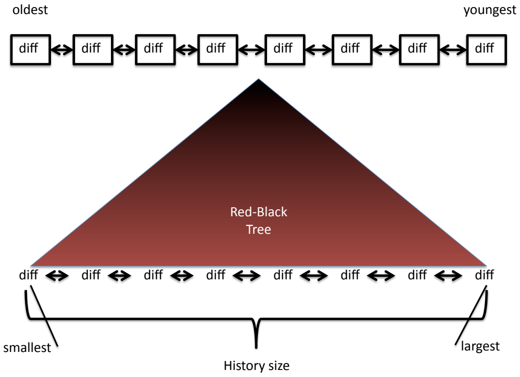

Figure 1. Data structures implementing QPRED. Doubly linked list holds fixed history of maximum differences. Red-black tree sorts current history of maximum differences.

of the 0.95 quantile of the current history of differences.

This implementation is simple and speed efficient. Each addition and deletion to the linked list is constant time, the addition and deletion of a value to the red-black tree is $O(log(H))$, and the scan for the quantile takes $(1 - q) \cdot H$ operations (if $q < 0.5$, and $(1-q) \cdot H$ operations if $q >= 0.5$ since the sorted list can either be scanned from largest to smallest or *vice versa*. The implementation is also space efficient since only the current list of historical entries is needed [3]. Note that the original QBETS prediction methodology on which this method is based includes a change-point detector that implements history trimming in the event that conditions change suddenly. Such an enhancement is possible for QPRED at the cost of additional predictor state and complexity. As our results indicate, however, for the current state of the practice with production private clouds represented in the traces we have examined, the additional complexity associated with change-point detection appears to be unwarranted.

In addition, we wish to concern ourselves not with the fraction of *time intervals* in which there is a delay but rather the fraction of instances themselves that experience a time delay at startup.

For example, suppose that the polling interval is 1000 seconds but that the instance interarrival time is 5000 seconds and that we want to maintain a probability of less than 0.05 of startup delay. In this case, we only want a delay once in every $20 \cdot 5000 = 100\,000$ seconds, so that only a fraction of 0.01 of the intervals should see a delay. Thus we must infer for the 0.99 quantile to account for the possibility of empty intervals.

More generally, suppose given a history of maximum occupancy numbers, a historic mean interarrival time I, polling interval t, and desired fraction α of jobs delayed at startup. At the beginning of each time interval:

- Calculate the target fraction $\beta = \min\left(\alpha, \frac{t}{I} \cdot \alpha\right)$ of time intervals experiencing a delay.
- Find a suitable upper bound M on the $(1 - \beta)$ quantile for the differences of the maximum occupancies.
- Adjust the number of hot spares so that there are a total M machines powered on above the maximum number occupied at any point in the previous time interval.

Note that the methodology adjusts the number of powered up or down at the beginning of each time interval. In practice, if the number of hot spares is inadequate during any interval, Eucalyptus will immediately initiate the power-up of a machine, but the VM requests that arrive before the machine is operational will be delayed.

Scheduling Methodologies

In Section III we compare the performance of four different scheduling methodologies. The performance of each methodology is characterized by the fraction of total power it uses, and the fraction of VMs that experience a start-up delay. The methodologies are defined as follows.

- **Power-greedy** – This scheduler results in the optimal power usage by a feasible implementation that considers VMs in the order they arrive (an $O(n)$ algorithm) without regard for the number of VMs that will experience a start-up delay. It uses a “greedy” selection strategy that always chooses a node that is in use and has sufficient capacity over one that is “empty” when making a VM assignment decision. It also keeps nodes powered off until they are needed and powers them off immediately when they become idle.
- **QPRED-greedy** – This scheduler makes greedy assignment decisions like Power-greedy, but it uses the quantile predictions to anticipate how many idle “hot spares” are needed at any moment to ensure that the probability a VM will be delayed falls below a target threshold.
- **Power-RR** – This algorithm is similar to Power-greedy in that it considers VMs in arrival order and only makes a single placement decision for each VM. However, instead of attempting to keep nodes “empty” so that they can be powered down, it uses a round-robin rule to assign VMs to nodes that are powered up when each VM arrives.
- **QPRED-RR** – Like Power-RR, this scheduling algorithm chooses among powered-up nodes when a VM arrives and must be assigned to a node. However, it uses the quantile predictions to anticipate the number of idle “hot spares” need to be available to meet a target VM delay probability threshold.

We modify the Eucalyptus VM scheduler to send a message to each node instructing it to put itself to sleep whenever that machine becomes idle. When the scheduler needs to start a VM, it consults an internal record of node

[3] The implementation actually maintains both the time-sorted linked list and the value-sorted red-black tree to improve speed efficiency. However, for the purposes of state exchange in the event of a fail-over, only the time-sorted list is needed – the red-black tree is reconstructed.

state and selects a node that currently has the capacity to run the VM and is also currently powered on. If no node is found, it then considers nodes that are in the process of "waking up" and chooses one that will have sufficient capacity once it is fully power on. Finally, if no "on" or "waking" nodes are located, it selects a node that is powered off, sends that node a *wake-on-lan* message [15] thereby putting it in the "waking" state, assigns the VM to the node, and waits until the node is fully powered up before starting it and any other VMs that are waiting. To keep VMs "packed" onto powered-up nodes, Power-greedy gives the nodes an arbitrary order and then always considers nodes in this order when making a placement decision. Power-RR is an alternative to Power-greedy that goes through each class of node ("on," "waking," and "off" in round-robin order (*i.e.* the scheduler starts with the next node in order when a new placement decision is needed).

The QPRED schedulers predict a bound on the maximum number of machines that will be required to start all VMs in a fixed time epoch such that the probability of a VM incurring a power-up delay is no greater than a fixed target probability supplied to the algorithm. QPRED-greedy uses the same greedy approach to making placement decisions as does Power-greedy, but it also attempts to power on enough hot spares (based on the quantile prediction) to control the probability that a future VM start will experience a start-up delay. Thus, compared to Power-greedy, QPRED-greedy trades additional speculative power usage for the ability to provide a statistically valid SLA. Alternatively, QPRED-RR is comparable to Power-RR except that it too uses the quantile prediction to forecast the number of additional nodes that must be powered up to meet a specific target SLA.

The difference between the greedy and round-robin versions of these schedulers is the degree to which the exploit multi-tenancy. The greedy schedulers will attempt to use a few machines as possible, thereby increasing the degree to which VMs will share nodes. As a result, they are more power-efficient than their round-robin counterparts; however, because of the greater potential sharing, VMs under a greedy schedule may experience greater I/O interference.

III. Experimental Results

The results presented in this section are generated from a faster-than-real time simulator that is able to "replay" each data set described in Table I using different scheduling methodologies. The simulator is able to replay each dataset as it was gathered (*i.e.* using the scheduling information in the data set). It also implements the different scheduling policies (both based on QPRED and otherwise), reports machine statistics such as node utilization, core utilization, power consumption, and the fraction of VMs that were required to wait for a node to power-up (also termed the "miss fraction" since the VM "missed" having an available node powered-on to start it). Because the results are simulated using datasets from machines that were not instrumented for power usage when the datasets were gathered, power consumption is reported as a fraction of the total power that was used. That is, the simulator records the ratio of time each node is powered-up using power-saving scheduler to the time that all node were powered up. We explain this method of measuring power consumption in greater detail in the next section.

We present data from six separate commercial private clouds – four implemented using Eucalyptus and two from "big data" companies that operate their own internal clouds using unspecified technology. All six clouds support the commercial activity of their operators (they are not operated for, *e.g.*, evaluation or investigative purposes). As a result, the supplies of these traces have made them available in anonymized form only to preserve both customer privacy and competetive market advantages.

The first data set (DS1) is taken from an organization with several large-scale software development efforts. While the private cloud is used for some company-wide service hosting, its primary use is to support software testing and development. DS1 captures private cloud VM activity that combines software development with service hosting, with an emphasis on development.

The second data set (DS2) is taken from an IT organization that "sells" time on a re-charge basis to other organizational units in its umbrella company. The accounting charges translate to operating budget for the following fiscal year, making the economic incentives similar to those driving a public cloud. Thus the usage of this cloud is not known (*i.e.*, the cloud does not have a specific purpose other than to host the workloads of its paying customers). The function of the umbrella company, however, makes it likely that much of the activity is generated by software development.

The third data set (DS3) is taken from a private cloud used to allow business partners to integrate their respective software products with the products made by the company operating the cloud. It also supports user and customer trials of the company's software products. Finally, these partners often use the cloud for demonstration or sales purposes. Thus the workload is a mixture of software development with on-demand hosting activities.

The fourth data set (DS4) comes from a cloud used exclusively for software development and testing at a software start-up company that uses an Agile [16] engineering process. The Agile process makes heavy use of testing during development so the workload in this data set represents a mixture of user-controlled VMs and VMs that are launched and terminated by an automatic testing system.

Finally, the fifth and sixth data sets (DS5 and DS6, respectively) are traces from "big data" open source frameworks (*e.g.* Apache YARN, Hadoop, and Apache Spark) that are operated by two moderate sized companies for their customers. Unlike DS1 through DS4, these traces contain only the framework parallelism and not the cloud provisioning information. Thus we must infer the cloud workload in terms of number of instances and cloud node count. We assume that each framework task requires a single core, and that the core count per machine is 8 in all cases. The node count is set to the maximum makespan (*i.e.* the maximum number of simultaneously executing tasks) shown over the duration of each trace.

While potentially optimistic in terms of the amount of parallelism that organization was able to exploit, these traces capture the scale, churn, and correlations that a cloud experiences when it is used to host "batch" big-data workloads. Many more VMs are instantiated and terminate simultaneously than in a cloud used for a variety of workloads (like those represented by DS1 through DS4). Further, DS5 and DS6 record longer time periods with a great deal more instance churn. DS5 contains approximately $41,800,000$ instances and DS6 contains $5,100,000$ instances respectively (compared to $1,000$ to $10,000$ for traces DS1 throigh DS4). In addition, the provider of the DS6 trace chose to obfuscate the specific dates recorde in the trace. The instance interarrival and lifetime durations are accurate as is the total duration of the trace (approximately 2.5 months) but the cloud operator chose to anonymize the precise starting time for the trace.

Table I provides summary descriptions of the cloud deployments from which we have gathered these data sets. All six data sets span several months of continuous usage. During the monitoring periods, many of the hosting organizations upgraded their respective clouds, in one case multiple times.

We begin by detailing the tradeoff between overall power usage and the probability that a VM's start will be delayed while a machine is powering up to host it. We compare Power-greedy, QPRED-greedy, Power-RR, and QPRED-RR in terms of both power usage and VM delay fraction. In what follows, we will use the term "miss fraction" interchangeably with the term "VM delay fraction" because from the perspective of a scheduler (particularly the predictive schedulers) a VM that experiences a VM start-up delay is a "miss" with respect to finding a machine powered on and ready to accept the VM.

Power Usage and VM Delay Fraction

Table II compares the performance of these four schedulers using the data set described in Subsection **??**. Each boldfaced number in the table denotes the fraction of maximal power that the scheduling methodology used. That is, the simulations compute the total number of node-seconds used for each data set as a hardware-independent measure of the power that would have been consumed in the absence of power-aware scheduling. The boldfaced numbers are the fraction of this maximal usage number for each data set (thus, *e.g.*, a lower fraction represents greater power savings).

The italicized numbers show the fraction of VMs that incurred a delay as a result of having to wait for a machine to reach a fully powered on state. For this experiment, we used a power-on interval of 600 seconds, taken to represent a typical amount of time it takes a server-class machine to start up, and a target delay probability of 0.05 for the SLA given to the user. Thus, when the predictor is accurate, the total fraction of VMs experiencing a delay for either QPRED method should be less than or equal to 0.05.

As an example, consider the results for DS3 in Table II. The boldfaced number in the second column (0.22) indicates the fraction of total power used with all of the machines powered up for the duration of the trace that Power-greedy scheduler would have used to complete the workload. Put another way, Power-greedy (which is the most power-efficient of the schedulers we examine) would use 22% of the power that was used by the system when it executed the workload originally, with all of its machines on and fully powered. At the same time, Power-greedy for DS3 generates a miss fraction of 0.27 (italicized number in column 2), indicating that 27% of the VMs would experience a start-up delay. In sum, Power-greedy for DS3 would use just 22% of the power that was used for the work load, but 27% of the user requests would incur a delay while waiting for a machine to power up.

In the third column for DS3, we show the power fraction (boldfaced) and VM delay fraction (italics) for QPRED-greedy. These data indicate that QPRED-greedy would have used 37% of the total power used originally, but only 2% of the VMs would have been delayed waiting for a machine to power up. Thus, QPRED-greedy would have used 15% more power (relative to the maximum) than Power-greedy while maintaining the 0.05 target probability (since 0.02 is less than 0.05) specified in the SLA.

Finally, in the fourth and fifth columns of the row for DS3, we show the results for Power-RR and QPRED-RR respectively. Power-RR uses 41% of the original power, but 60% of the VMs experience a start-up delay. Meanwhile, QPRED-RR uses 59% of the original power (18% more that Power-RR relatively speaking) while respecting the 0.05 miss fraction specified in the SLA ($0.02 < 0.05$).

Predictor Efficacy

The data in Table II used an SLA with a target VM delay probability of 0.05 for all experiments. As described in Section II, the predictor used in both QPRED schedulers attempts to estimate the quantile of the distribution of the maximum number of nodes occupied during each time epoch corresponding to this target probability. Thus for a target probability of 0.05, the quantile estimator attempts to choose the number of nodes that correspond to the 0.95 quantile of the distribution of the maximum node occupancies across epochs. From the table, the predictor is correct for a 0.05

Data Set	Nodes	Cores/Node	Time Period	Description
DS1	13	24	Aug. 2012 to Oct. 2012	Large company with 50,000 to 100,000 employees
DS2	7	12	Aug. 2012 to Apr. 2013	Medium sized company with 2,000 to 5,000 employees
DS3	7	8	Aug. 2012 to May 2013	Small company with 50 to 100 employees
DS4	12	8	May 2013 to Sep. 2013	Start-up company with 5 to 10 employees
DS5	580	8	May 2015 to Feb. 2016	Open Source “Big Data” company with 50 to 100 employees
DS6	1169	8	2.5 months (*actual dates obfuscated*)	Internet marketing company with 150 to 200 employees

Table I
Summary of Private Cloud Dataset Characteristics

Data Set	Power-greedy	QPRED-greedy	Power-RR	QPRED-RR
DS1	**0.56** *0.08*	**0.62** *0.02*	**0.92** *0.06*	**0.87** *0.00*
DS2	**0.33** *0.45*	**0.51** *0.05*	**0.76** *0.20*	**0.83** *0.01*
DS3	**0.22** *0.27*	**0.37** *0.02*	**0.41** *0.60*	**0.59** *0.02*
DS4	**0.35** *0.46*	**0.56** *0.04*	**0.43** *0.67*	**0.67** *0.01*
DS5	**0.40** *0.61*	**0.60** *0.01*	**0.93** *0.18*	**0.98** *0.00*
DS6	**0.48** *0.80*	**0.69** *0.02*	**0.97** *0.07*	**0.98** *0.01*

Table II
Comparison of Scheduler performance. Boldfaced numbers are fraction of maximal power. Italicized numbers are fraction of VM’s delayed. QPRED target delay fraction is *0.05*

VM delay probability since all of the observed delay fractions are less than or equal to 0.05.

In Table III we show the VM delay fraction for QPRED-greedy that results from parameterizing the predictor with different target quantiles corresponding to different SLAs.

In each experiment, we use an epoch interval of 1000 seconds and a power-up delay of 600 seconds (the same as for the results in Table II). In each column except the first we show the fraction of original power usage in boldfaced type and the miss fraction in italics for the target quantile q shown in the first row. We underline the entries where the observed VM delay fraction is greater than the target quantile (*i.e.* an SLA violation) indicating that the predictor failed to achieve a conservative bound.

As expected, the fraction of maximal power increases as the target quantile decreases. That is, smaller the fraction of VMs that can miss according to the SLA, the more power the cloud must use to ensure that the SLA is met. For example, for DS4, an SLA of 0.01 uses 65% of the original power. If an SLA of 0.25 is chosen, the true miss fraction rises to 0.11 but the cloud uses only 45% of the original power. Thus the price of a 0.01 SLA guarantee versus a 0.25 SLA guarantee is 20% in terms of power usage for DS4.

For all target quantiles except $q = 0.01$ for DS2 the predictor’s bound on VM delay fraction holds, although it appears quite conservative in many cases (*e.g.* the VM delay fraction for DS1 is 0.07 for a target of 0.25). The predictor misses outright with a miss fraction of 0.03 for DS2 with a target quantile of $q = 0.01$, however. This failure illustrates the effect that autocorrelation in the interarrival time series can have on our methodology. Specifically, the DS2 data set contains periods of time when few VM starts occur and also short intervals when a large number of VMs arrive. The prediction methodology does not take this “burstiness” into account. Thus, when a burst of VMs occurs in the DS2 data set, QPRED-greedy does not have enough machines ready and idle to absorb the burst such that at most only 1% of the VMs will experience a delay per the terms of the SLA.

To investigate the effect spin-up delay has on the results, Table IV shows the VM delay fractions using QPRED-greedy with a target quantile of 0.05 and different machine spin-up delays. For this experiment, we show the results for Power-greedy and vary the spin-up delay from 60 seconds to 1800 seconds while keeping the length of the time epoch and the history length both at 1000 (as they were in the previous experiments) and the target VM delay probability set at 0.05. The miss fractions are shown in italics and fractions that exceed the target SLA probability of 0.05 are underlined. From this information, it is clear that miss fraction increases with spin-up delay; however, the rate of increase is slow.

In [11] and [10], the authors report that the savings benefits gained by powering down machines when they are not needed can be overshadowed by the use of additional “peak” power during the spin up phase. When a machine is powered on, it may use more power (*e.g.*, to accelerate disks to operational speed) relative to its steady-state or idle-state usage. The ratio of peak usage during start-up to steady-state usage varies by machine manufacturer and model as well as by configuration (*e.g.*, the number and type of disks attached). The authors of both works note that the additional usage during power-up can be as much as 60% more than steady-state.

Data	q=0.01	q=0.05	q=0.10	q=0.15	q=0.20	q=0.25
DS1	**0.68** *0.01*	**0.62** *0.02*	**0.58** *0.03*	**0.57** *0.04*	**0.55** *0.07*	**0.55** *0.07*
DS2	**0.53** *0.03*	**0.51** *0.05*	**0.52** *0.05*	**0.47** *0.08*	**0.46** *0.11*	**0.45** *0.15*
DS3	**0.45** *0.01*	**0.37** *0.02*	**0.36** *0.03*	**0.35** *0.03*	**0.34** *0.04*	**0.32** *0.05*
DS4	**0.65** *0.01*	**0.56** *0.04*	**0.52** *0.04*	**0.49** *0.06*	**0.47** *0.07*	**0.45** *0.11*
DS5	**0.68** *0.01*	**0.58** *0.01*	**0.56** *0.01*	**0.53** *0.01*	**0.51** *0.01*	**0.50** *0.01*
DS6	**0.80** *0.01*	**0.69** *0.02*	**0.63** *0.04*	**0.60** *0.06*	**0.58** *0.09*	**0.56** *0.10*

Table III

Power usage fraction in boldface and VM delay fraction in italics for different target quantiles using QPRED-greedy.

Data	60s	90s	120s	300s	600s	900s	1200s	1800s
DS1	*0.01*	*0.01*	*0.01*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*
DS2	*0.02*	*0.02*	*0.02*	*0.04*	*0.05*	*0.06*	*0.05*	*0.06*
DS3	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*
DS4	*0.03*	*0.03*	*0.03*	*0.03*	*0.04*	*0.04*	*0.04*	*0.04*
DS5	*0.01*	*0.01*	*0.01*	*0.01*	*0.01*	*0.01*	*0.01*	*0.01*
DS6	*0.01*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*	*0.02*	*0.03*

Table IV

VM miss fraction only QPRED-greedy and target quantile of 0.05 as a function of increasing VM spin-up delays.

In Table V we show the power savings for QPRED-greedy over a range of peak-to-steady state ratios. For these experiments, we use a target quantile of 0.05 and a spin-up delay of 600 seconds. A ratio of 1.0 shows the case when there is no difference between power consumption during spin-up and steady-state. Thus column 2 of Table V (marked as 1.0) corresponds to column 3 (marked as $q = 0.05$) of Table III discussed previously. Even if the spin-up cost were five times steady-state, the additional power usage is no more than 1% with a spin-up time of 600 seconds.

For the general cloud workload traces, these results do not contradict the previously published work in [11] and [10]. Rather, they indicate that with a target SLA quantile of 0.05 and production cloud workloads, QPRED-greedy does not generate enough spin-up events for a large peak-to-steady state ratio to have a substantial effect on power savings.

However for the big-data batch workload traces, where machines starts and stops are highly correlated, the surge power can be significant. Notice that QPRED correctly predicts the need for additional machines (*cf* Table IV). However because so many machines may be needed at one time to service the batch workloads, the additional power required for spin up can be significant when it is a large multiple of the steady-state power consumption *and* the spin-up time is long.

IV. Related Work

Both because clouds aggregate usage and also because they commoditize compute and storage capabilities, they are especially well-suited for the implementation of automatic power optimization. In [10], [9] and [17] the authors discuss the efficacy of various power level formulations' for data-center-hosted processors. This work, like ours, involves the application of Markov/time-series methods to the problem of power management. In these papers the Markovian approach appears in service of an $M/M/k$ queuing model. Our work, which focuses on clouds rather than data centers, also uses a Markov-based approach to workload but in a much more direct way: we use a fast algorithm to make sample-based estimates of confidence bounds on transition probabilities (indeed with some further simplifying hypotheses). While their use of time series is in some sense more sophisticated than ours, we have found that for our purposes nonparametric and model-agnostic methods yield better results. Additionally, our work examines the SLA that a cloud must provide with respect to VM start-up delay; their work (perhaps because it focuses on workloads in data centers, where start-up delay is not typically subject to an SLA) does not consider start-up delay guarantees.

In [18] the authors formulate the problem in terms of multi-dimensional optimization and then explore a set of heuristics for improving power usage. Our efforts focus on predictive enhancements that augment cloud schedulers used in production today.

The work in [14] investigates the power efficiency of the same scheduling strategy that has been implemented by Eucalyptus as the Power-greedy scheduler. In addition, they explore the effects of additional "hot spares" (called a "pool") in this work. As described, our work prioritizes user experience in the form of an SLA and uses an on-line predictive methodology to predict how many hot spares are needed. Because of the similarity in base-line schedulers between OpenNebula [19] (the test platform for this work) and Eucalyptus, however, our approach should be directly applicable to their test environment.

In [20] and [21] the authors use a variety of statistical techniques including time series analysis and clustering to predict VM workloads from a virtualized data center that is intended to be used as a private cloud. Their study uses CPU utilization data gathered from each VM across a history of time intervals to predict aggregate load in the next time interval. Our work is similar in that we too discretize time into epochs and use time series of measurements to make a prediction for each epoch immediately before it begins. However, our methods uses measurements of overall cloud load rather than an aggregation of VM CPU utilization. Further, our approach predicts quantiles as a way of implementing user-facing SLAs whereas their method generates point-value predictions.

V. Conclusion and Future Work

This work shows that it is possible to use a simple, computationally efficient prediction methodology based on quantile estimation to improve cloud power usage while also implementing an SLA governing machine virtual machine start-up delay. The methodology predicts a conservative bound on the number of machines that must be powered on at any moment to ensure that the probability of having to power up a machine (*i.e.*, a miss) is at or below the target

Data Set	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
DS1	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62	0.62
DS2	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.51	0.52
DS3	0.37	0.37	0.37	0.38	0.38	0.38	0.38	0.38	0.38
DS4	0.55	0.56	0.56	0.56	0.56	0.56	0.56	0.56	0.56
DS5	0.60	0.61	0.62	0.63	0.64	0.65	0.66	0.68	0.69
DS6	0.69	0.70	0.71	0.71	0.72	0.73	0.73	0.74	0.75

Table V

VM POWER USAGE FRACTION FOR QPRED-GREEDY, TARGET QUANTILE OF 0.05, AND SPIN-UP DELAY OF 600 SECONDS AS A FUNCTION OF INCREASING PEAK POWER RATIO DURING SPIN-UP.

set by the cloud administrator. We illustrate the efficacy of the approach using VM activity traces gathered from four enterprise private clouds that were in production use at the time of their instrumentation. Our results show that QPRED (which is non-parametric and both computationally and space efficient) generates substantial power savings under settable probabilistic constraints on the tradeoff between power savings and degraded user experience.

References

[1] J. Murty, *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*. O'Reilly Media, Inc., 2009.

[2] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? an architectural map of the cloud landscape," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE Computer Society, 2009, pp. 23–31.

[3] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on*. IEEE, 2009, pp. 124–131.

[4] Eucalyptus Systems Inc. (2013) http://www.eucalyptus.com.

[5] K. Pepple, *Deploying OpenStack*. O'Reilly, 2011.

[6] Apache Cloudstack. (2013) http://cloudstack.apache.org.

[7] D. Nurmi, J. Brevik, and R. Wolski, "Qbets: Queue bounds estimation from time series," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2008, pp. 76–101.

[8] D. Nurmi, R. Wolski, and J. Brevik, "Probabilistic advanced reservations for batch-scheduled parallel machines," in *Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming*. ACM, 2008, pp. 289–290.

[9] A. Gandhi, "Dynamic server provisioning for data center power management," Ph.D. dissertation, Intel, 2013.

[10] A. Gandhi, M. Harchol-Balter, and I. Adan, "Server farms with setup costs," *Performance Evaluation*, vol. 67, no. 11, pp. 1123–1138, 2010.

[11] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE computer*, vol. 40, no. 12, pp. 33–37, 2007.

[12] H. N. Van, F. D. Tran, and J.-M. Menaud, "Performance and power management for cloud infrastructures," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 329–336.

[13] R. Bahsoon, "A framework for dynamic self-optimization of power and dependability requirements in green cloud architectures," in *Software Architecture*. Springer, 2010, pp. 510–514.

[14] A. J. Younge, G. Von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, "Efficient resource management for cloud computing environments," in *Green Computing Conference, 2010 International*. IEEE, 2010, pp. 357–364.

[15] wake-on lan. (2013) http://en.wikipedia.org/wiki/Wake-on-LAN.

[16] A. Software Development. (2013) http://en.wikipedia.org/wiki/Agile_software_development.

[17] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu, "Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 571–580.

[18] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 826–831.

[19] J. Fontán, T. Vázquez, L. Gonzalez, R. S. Montero, and I. Llorente, "Opennebula: The open source virtual machine manager for cluster computing," in *Open Source Grid and Cluster Software Conference*, vol. 86, 2008.

[20] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 1287–1294.

[21] R. Birke, L. Y. Chen, and E. Smirni, "Data centers in the cloud: A large scale performance study," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012, pp. 336–343.