Using Trustworthy Simulation to Engineer Cloud Schedulers

Alexander Pucher, Emre Gul, Rich Wolski, and Chandra Krintz
Department of Computer Science
University of California, Santa Barbara
{pucher, emre, rich, ckrintz}@cs.ucsb.edu

***Abstract*—In recent years, researchers have contributed promising new techniques for allocating cloud resources in more robust, efficient, and ecologically sustainable ways. Unfortunately, the wide-spread use of these techniques in production systems has, to date, remained elusive. One reason for this is that the state of the art for investigating these innovations at scale often relies solely on model-driven simulation. Production-grade cloud software, however, demands certainty and precision for development and business planning that only comes from validating simulation against empirical observation.**

In this work, we take an alternative approach to facilitating cloud research and engineering in order to transition innovations to production deployment faster. In particular, we present a new methodology that complements existing model-driven simulation with platform-specific and statistically trustworthy results. We simulate systems at scales and on time frames that are testable, and then, based on the statistical validation of these simulations, investigate scenarios beyond those feasibly observable in practice. We demonstrate the approach by developing an energy-aware cloud scheduler and evaluating it using production and synthetic traces in faster than real time. Our results show that we can accurately simulate a production IaaS system, ease capacity planning, and expedite the reliable development of its components and extensions.

I. Introduction

Cloud computing, in the form of Infrastructure as a Service (IaaS), has emerged as a new paradigm for Information Technology (IT) management of data center infrastructure. Under the IaaS cloud model, users request that data center resources be *provisioned* for their exclusive use via network-facing web service interfaces (APIs). "The Cloud" advertises data center resources as commodities, each described by a Service Level Agreement (SLA). Thus, cloud users provision capacity that meets SLA guarantees rather than specific resources. When capacity is no longer needed, the cloud APIs allow users to decommission the resources they have previously provisioned and return them to the pool of commodities that are available for general use. Often, some or all of the resources that are available are "virtualized" making it possible for the cloud to isolate under software control each user's allocation, and to share physical resources across users. Public clouds implement this paradigm on a "for-fee" lease basis in which each user is charged an advertised rate for occupancy based on either time (*e.g.* rental of virtual machines) or space (*e.g.* storage occupancy). Private clouds implement the same model using accounting and quotas instead of for-fee leases within a single "private" organization.

Because IaaS clouds allow automated, self-service user provisioning of data center resources, they can reduce the labor burden associated with both software development and IT management, sometimes dramatically. For this reason, cloud computing is currently the subject of considerable commercial interest and investment. However, to achieve the optimization benefits promised by the model, clouds must scale, both in terms of the resources they manage and the number of simultaneous users they can support. Moreover, for cloud users, the IaaS platform plays the role that operating systems play for individual machines: it is an abstraction layer that must function as if it is part of "the hardware." In particular, when the cloud fails, it is as if the hardware has failed leaving users with no remedial options other than to wait for service to be restored. Thus an IaaS platform must implement fully automated provisioning at the greatest possible user and resource scales with the reliability at least as high as that of a single-machine operating system.

These requirements make IaaS research and experimentation particularly challenging. Production-quality IaaS platforms are complicated, highly scalable, reliable distributed systems. Public cloud service providers such as Amazon AWS [1], Google Cloud Platform [2], IBM SoftLayer [3] and Rackspace Cloud [4] hold the implementation details associated with their respective platforms as trade secrets. However private cloud platforms such as Eucalyptus [5], [6], OpenStack [7], and CloudStack [8] implement some or all of the same functionality at the data center level as production-quality, open source.

Thus, it is possible to study new optimization techniques for cloud computing using production-quality systems by leveraging these private cloud technologies. To do so, however, the research must take care not to violate the performance, scale, or reliability constraints already engineered into the platform. Indeed, the stewards of these open source projects often reject code contributions that appear to function in prototype form, but ultimately fail when subjected to production quality-assurance testing.

In this paper, we outline a process for conducting cloud scheduler research for private clouds that takes into account the ultimate need for the engineering of a reliable, efficient, and accurate implementation as part of a production-quality platform. Cloud schedulers are central to the optimization gains that the cloud makes possible. For this reason, cloud administrators are reluctant to change them without convincing evidence that a new scheduler will not negatively impact reliability and stability, and that substantive benefit will result from the adoption of a new optimizing scheduler. Similarly,

IaaS development engineers need to be able to trust that a feature, which performs well in prototype form, can be implemented subject to the reliability and scale requirements associated with production computing. Thus our methodology is designed to engender the trust necessary both, to ensure IT adoption and also to justify the engineering effort required to achieve production levels of performance and reliability for cloud schedulers.

Fundamentally, our approach describes a process for developing a cloud scheduler from research conception to production software development through the use of *validated simulation*. For cloud computing, simulation systems to date [9], [10], [11], [12], [13], [14] focus on *ab initio* techniques in which various low-level cloud components (machines, networks, storage devices, etc.) are simulated and these component simulations are then composed efficiently into a full system simulation. This "bottom up" approach is both flexible and easily extensible, and yields insights that stem from comparative ranking (*e.g.* "this" configuration is better than "that" one). The value of this approach cannot be underestimated, however, the scale and reliability requirements for clouds present challenges for *ab initio* methods with respect to accuracy that must be addressed before they can be considered "trustworthy" from an engineering perspective. This approach tends to produce results at scales that are difficult to test empirically, and even if evaluated on a public cloud, the component models may be inaccurate because public providers do not reveal their implementation details. Finally, the composed cloud model may become so complex that the error interactions between component models become untamable.

Our work explores an alternative approach rooted in *perturbation theory* [15] that focuses on validation of simulated results against empirical measurement (at the cost of flexibility and extensibility) as a way of addressing the engineering needs that cloud developers and practitioners have. Specifically, we build a parsimonious "top down" model of the end-to-end system that derives from the implementation specifics of the system. We then add "noise" (taken from statistically sampled empirical measurements of the system) to "perturb" this model. For validation, we analyze the perturbed model's outputs statistically over repeated runs (*i.e.* via a Monte-Carlo [16] approach) and compare them to distributions of measurements taken from repeated runs of the end-to-end system. For cloud schedulers in particular, this approach has proved fruitful because the models are quite parsimonious (reducing the possibility of error propagation) and the system measurements are easily gathered at scales that are feasible for repeated measurement. Once validated, the model can then be scaled up in any dimension characterized by independent performance response. For example, if the performance of the physical machines hosting user-allocated virtual machines is independent (due to the isolation properties of the cloud platform) then the physical machine count can be scaled without introducing additional error.

We emphasize that our work is intended to complement *ab initio* approaches in that it targets the development of a specific component (schedulers in our case), that the component must be amenable to a perturbation-based approach to modeling, and that scaling is trustworthy only in the dimensions of independence. Further, our approach is intended to produce accuracy only in the parameters that are necessary for a particular component's operation *as an isolated feature*. That is, the method is appropriate for clouds because (for reliability reasons) component operation is isolated through internal modularity techniques. Our method relies on this engineering property, which is common to the cloud platforms, and access to the source code so that the relevant parameters can be identified.

Even with these restrictions it is possible to use our simulation technique to explore scaling properties, etc. in a manner similar to previous approaches. A key additional benefit of our method is that the results are validated at scales that can be tested and that there is evidence that their accuracy is preserved at enterprise scales.

This paper describes our methodology in high-level terms and illustrates its use to develop a new, power optimizing scheduler for Eucalyptus [5], [6], a popular, open source private cloud used for production by many commercial enterprises. We describe the model, its discrete-event simulation, and its validation against a working implementation. We also describe and detail the use of the simulation in various enterprise capacity-planning contexts using production cloud traces as a way of demonstrating its utility.

In summary, this paper makes the following contributions:

- We outline a simulation approach that is designed to support production-quality engineering of cloud platforms by applying a new approach – perturbation-based modeling – to cloud simulation.
- We demonstrate the use of this methodology in the implementation of a new, power optimizing scheduler for a production-quality private cloud platform.
- We evaluate the simulator's utility for capacity planning in both, backtesting and forward-looking planning using synthetic workloads and recorded load traces gathered from production private clouds.

In the sections that follow, we present our simulation approach. We first describe the steps of our methodology for top-down model development, discrete event simulation, model fitting, and validation. As an example of this process, we show how we use it to implement and evaluate a new power-optimizing scheduler. We then present an empirical evaluation of our approach that includes registration of our simulator against a production-quality Eucalyptus private cloud, an evaluation of the scheduler, and an investigation into capacity planning use cases using the system. We then present related work and conclude.

II. Methodology

We outline a process for conducting cloud scheduler research for private clouds in terms of a specific example in which we seek an implementation of a new power-optimizing scheduler for a private cloud. The scheduler uses on-line machine learning methodologies to predict (in real time) when machines should be powered on and off to avoid delays associated with machine spin up. Before an expensive engineering effort can be launched to implement such a scheduler or a skeptical IT professional can be convinced to introduce a

new methodology, the reliability, performance, and efficacy properties of a new scheduler must be verified. Our goal with this process is to facilitate accurate, faster-than-realtime, end-to-end testing via validated simulation.

1) Top-Down Approach: The goal of the methodology is to use an, alternative, "top down" approach to simulation that models only those parameters that are necessary to capture the behavior of the component of interest with sufficient accuracy. Identifying the parameters of this model requires an understanding of the fault isolation properties of the platform which, in our example use case, comes from source code inspection. The fault isolation properties establish the independence of our model parameters which is required for trustworthy scaling of our simulations.

The approach is to:

1) start with the most parsimonious model of end-to-end behavior that is possible,
2) perturb the model using statistical sampling technique to represent unmodeled behavior,
3) test the model by comparing its outputs generated in simulation to measurements taken from the "real world" system,
4) if the model is insufficiently accurate, add terms, adjust the perturbation, and repeat.

Thus every addition of a variable to our model of the cloud should be justified by a necessary increase in accuracy. Variables that only contribute marginally to the aggregate result are omitted and modeled in aggregate as "perturbing" error terms. The level of accuracy that is acceptable is ultimately decided by the consumers of the simulation. In an engineering context the error terms may serve as inputs to a risk analysis, where variability is acceptably low when the difference in risk that greater accuracy would engender is deemed insignificant by those taking the risk.

2) Developing the Model: The first step in our approach is a white-box inspection of the documentation and source code. With information about the control and data flow in hand, we are able to identify critical inputs, cloud components and their interactions, and relevant output metrics.

In this research, we are interested in evaluating a new cloud scheduler, which requires user requests, the physical platform configuration, and the allocation algorithm as inputs. The cloud model consists only of a set of independent nodes with fixed resource capacities that hold a number of instances with fixed requirements. Interactions between this model and the scheduler take place when a request arrives or the life time of an instance expires. Otherwise, scheduler and model are isolated.

The outputs of the cloud scheduler that we can observe are (i) request acceptance rate and (ii) the allocation of "virtual machines" (VMs) to nodes over time. We quantify this behavior by computing the aggregate CPU time for each physical node devoted to work assigned to it by the scheduler. Comparing node CPU time, both in simulation and actual measurements, succinctly captures the end-to-end behavior of the system under test for the cloud scheduler component.

Note that a new scheduler may require additional modeling terms beyond those that capture the existing system's behavior. In our case, we wish to implement and test a power-aware scheduler that predictively and pro-actively powers on and off nodes based on recent load history [17]. To enable this, we must extend the model to represent periodic polling of load (the periods are called "epochs" in the scheduler algorithm) and power states of the nodes (*awake*, *waking*, and *asleep*) that the scheduler can manipulate via messages to the nodes. We extend the output set of this scheduler to include the aggregate power-up delay it generates and the amount of time each node spends in the *awake* or *waking* state. We perturb the model by representing the delay necessary to power a node up as empirically determined distributions (so as to avoid their simulation overhead).

3) Discrete Event Simulation: To simulate the system in faster-than-real time, the next step is to develop a discrete-event simulation that captures only the changes in the states specified in the model. In our example, scheduler events are triggered by

- the arrival of a new VM request from the input trace
- the acceptance and launch of a new VM assigned to a node
- the termination and cleanup of a VM as reported by the node running it
- the expiration of a timer marking epoch boundaries,
- the expiration of a timer marking the end of a node power-up sequence

The simulation of the scheduler (either the existing or the new power-optimizing scheduler) from these events takes a trace of VM activity, which we represent as start-time and and duration pairs for a set of VMs.

Note that this event list demonstrates the parsimony in our approach. Through inspection, it is clear that Eucalyptus breaks the VM start and termination sequence into a series of separate "phases" for the purpose of error handling and fault tolerance. We represent these in our simulator by the perturbation of VM start-up and termination delays. Notice also that we can omit the node power-down time as its addition does not change the results in a way that we could detect.

At a high level, the simulation works as follows. User requests consist of request time (arrival), instance lifetime (duration) and instance type (size). The platform configuration contains physical node IDs and capacity (cores, memory, disk). The scheduler assigns requested instances to nodes, and removes them as they expire based on a policy (the algorithm implemented). Additionally, the scheduler is notified when node power states or epoch times change in order to perform power-budget accounting.

4) Measurements: To fit and evaluate the simulation model we extract performance information from a live cloud. We require two types of measurements: those we use to introduce perturbations (e.g. VM start-up, termination, etc.) and those that we use to validate the simulations (*i.e.* the aggregated outputs of the scheduler).

To collect these measurements, we use a combination of log analysis and instrumentation. Log analysis is preferable since it avoids the possibility of disturbing system performance

through the introduction of instrumentation. In the case where the existing logs do not carry the information with sufficient resolution to drive the simulation, we take care to modify the source code of the platform to introduce additional logging information in a way that is unlikely to change execution performance. For example, logging new events that require synchronization of otherwise asynchronous activities must be avoided.

5) Scheduler Operation: Both, the existing scheduler and the power-optimizing scheduler must be implemented for the simulator. The accuracy risk (and one of the reasons necessitating validation) comes from the observation that the simulated and real implementations may differ. Ideally, both the discrete-event simulation and the implementation for the real system share the same source code. In our example, that sharing is possible, but we opted instead to rely on validation so that we might implement the schedulers in different programming languages. The production system schedulers are written in C and our discrete event simulation is written in Scala.

The existing production scheduler uses a "greedy' scheduling algorithm to maximize multi-tenancy. When a new VM is to be assigned to a node, the scheduler considers the node list in a fixed order and uses a first-fit assignment algorithm.

The power-optimizing scheduler uses load measurements taken over discrete epochs to predict how many powered-up machines will be needed in the "next" epoch to avoid a power-on event with a specified probability. While a node is being powered on, the VM start will be delayed by the remaining duration of the power-up sequence. This delay is experienced by the user directly. Thus the goal of the power-optimizing scheduler is to minimize power usage, subject to an SLA specified by the cloud administrator that limits the probability of any given user experiencing a power-up delay. As in the greedy scheduler, we order hosts by status (*awake*, *waking*, and *asleep*) and ID. We place an incoming VM on the first available *awake* host (followed by a *waking* host). If no powered-up host can be found, the request will be enqueued for a powered-down node, which is immediately sent a wake-on-lan message. A start delay is incurred whenever a VM is placed on a machine in *waking* or *asleep* state.

The power manager uses a fast, non-parametric quantile predictor and makes conservative estimates about the number of hot spares needed to fulfill the responsiveness (non-delay) SLA. In fixed time steps - epochs - the current utilization of the cluster and the size of request bursts in the past is used to determine this target count of active nodes. Depending on this target count and the current state of the system, additional nodes are then woken up or powered down.

III. Results

In this section, we evaluate our approach and its example implementation. We first overview our experimental setup and then present the results that we achieve by statistically registering our simulator with the actual target IaaS system it simulates. Using registered simulation, we then evaluate our power-aware scheduler and evaluate a number of different capacity planning scenarios, using a number of different traces (actual and synthetic) and cloud configurations.

TABLE I: Summary of Synthetic Workloads. Units are in Seconds.

Name	Total Duration	VM Count	Arrival	Duration
Exponential	36305	443	$\lambda = 0.0125$	$\lambda = 0.002$
LogNormal	35646	420	$\mu = 3.8$ $\sigma = 1.0$	$\mu = 4.5$ $\sigma = 1.0$

TABLE II: Summary of Empirical Cluster Attributes collected

Attribute	Description
VM start delay	Instance start delay until boot sequence
VM teardown delay	Instance termination delay until resources freed
Node wakeup delay	Time required for node wake-on-lan

A. Experimental Setup

For our empirical measurements we use a seven node commodity hardware cluster. Each node runs on CentOS v6.5 and holds four cores, 8 GB ram, and a 500 GB hard drive and is connected to the network via two 1 Gbit ethernet links. We set up Eucalyptus v3.4.2 with a dedicated head and storage node and six nodes serving as instance hosts. Since we need control over the placement of instances and power management of nodes, we install from source and inject a small code modification that enables explicit node selection by our scheduler. We implement the power manager to interact with the cluster controller via its shared-memory interface. Eucalyptus is a production-quality system and as such includes a number of security features that are in place to prevent these kinds of outside modifications. For this reason, we temporarily disable message signature verification to make the injection of load traces less labor intensive to implement.

We use a number of synthetic workloads and production cluster traces [18] to evaluate the simulator and the power-optimizing scheduler. The synthetic workloads are generated from exponential and lognormal distributions for instance arrival times and durations (details in Table I). We also have access to anonymized traces from Eucalyptus installations used in enterprise production (described later in Table VII). Using these traces it is possible to "replay" the VM load and scheduling activity that took place when they were gathered, either in simulation or on a working Eucalyptus system.

B. Simulation Registration

For registration we execute a benchmark trace on a single node which has been separated from the six node Eucalyptus IaaS cluster. The trace contains 100 uniform instance start- and stop-requests over a period of 10 hours. We collect the empirical samples of instance startup, instance termination, and power-up delays. The specific attributes that we profile in this study are shown in Table II.

Eucalyptus uses a polling model so that it can control message traffic internally. Thus, the nodes do not report these events to the rest of the system until they are polled. In order to reduce modes in the observed distribution due to fixed polling delays we introduce a small random variation in start- and stop-times of instance requests. Separately, the latencies for hibernation are obtained by manual execution of a script power cycling the machine. We then configure the simulator to use

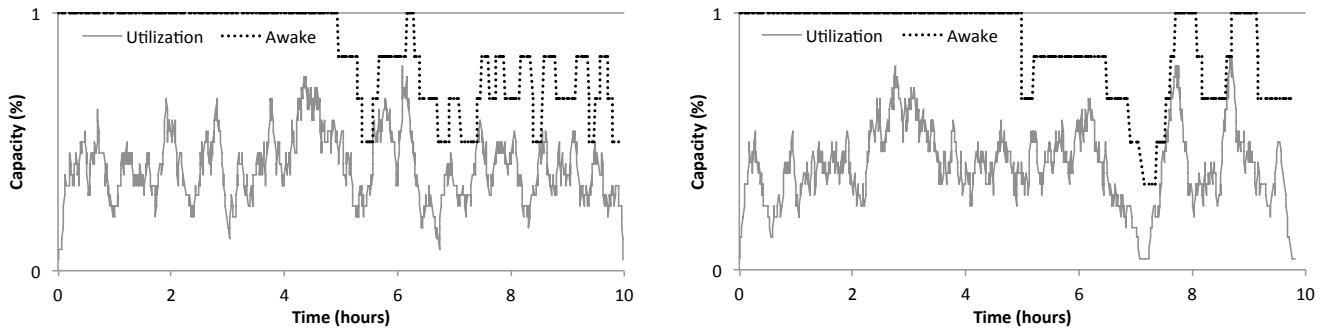

Fig. 1: Timeries showing synthetic exponential (left) and lognormal (right) workload trace with power-optimizing scheduler activated at the 5 hour mark. The x-axis depicts time in one hour intervals, and the y-axis shows the fraction of the number of cores occupied. The dotted line shows the fraction of cores that belong to nodes that are powered-up.

TABLE III: Utilization per Node (Exponential Trace)

	All	A	B	C	D	E	F
sim (mean)	0.4008	0.8727	0.7564	0.5195	0.2217	0.0346	0.0000
sim (sd)	0.0066	0.0033	0.0091	0.0085	0.0085	0.0047	0.0000
real (mean)	0.4033	0.8742	0.7551	0.5250	0.2311	0.0344	0.0000
real (sd)	0.0052	0.0053	0.0061	0.0062	0.0075	0.0024	0.0000

TABLE IV: Uptime per Node (Exponential Trace)

	All	A	B	C	D	E	F
sim (mean)	0.8711	1.0000	1.0000	1.0000	0.9128	0.7766	0.5375
sim (sd)	0.0127	0.0000	0.0000	0.0000	0.0166	0.0235	0.0119
real (mean)	0.8758	1.0000	1.0000	1.0000	0.9312	0.7704	0.5529
real (sd)	0.0094	0.0000	0.0000	0.0000	0.0115	0.0174	0.0098

TABLE V: Utilization per Node (Lognormal trace)

	All	A	B	C	D	E	F
sim (mean)	0.3974	0.8555	0.7305	0.5140	0.1960	0.0665	0.0217
sim (sd)	0.0045	0.0024	0.0053	0.0082	0.0039	0.0001	0.0023
real (mean)	0.3985	0.8550	0.7223	0.5213	0.2025	0.0696	0.0202
real (sd)	0.0043	0.0022	0.0046	0.0059	0.0050	0.0037	0.0036

TABLE VI: Uptime per Node (Lognormal trace)

	All	A	B	C	D	E	F
sim (mean)	0.8565	0.9851	0.9851	0.9637	0.9161	0.7192	0.5696
sim (sd)	0.0025	0.0000	0.0000	0.0020	0.0000	0.0042	0.0038
real (mean)	0.8605	0.9851	0.9851	0.9652	0.9178	0.7292	0.5805
real (sd)	0.0048	0.0000	0.0000	0.0000	0.0001	0.0112	0.0033

these empirical latency distributions and prepare for testing the power manager with synthetic workloads.

We use synthetic workloads during the registration phase so that we can ensure that the observed response of the system is meaningful on a feasible time frame. That is, a replay of the production traces described in Subsection III-A in real time would span months. Alternatively, selective extractions of tractable "busy" periods might skew the sample and the attempt to "speed up" the trace (*i.e.* using a fitted probability model as described in [19]) could introduce additional error.

Thus we choose two synthetic traces each having a duration of 10 hours, with a mean utilization of 1/3 of the 6 node cluster capacity. The first trace is generated from an exponential distribution for arrival times and instance durations, whereas the second trace uses a lognormal distribution for both. For the exponential distribution, these values of λ correspond to a mean inter arrival time of 80 seconds and a mean duration of 500 seconds. For the lognormal distribution, the mean inter arrival time is 81 seconds and the mean duration is 785 seconds. Note that there is a minimum lifetime of 360 seconds to allow for instance startup and all VM requests issued are single-core and uniform in memory and disk requirements.

In all test cases the power-optimizing scheduler is configured to guarantee a responsiveness SLA that at least 95% of all start requests will not be affected by a delay due to waking a node from hibernation. The epoch length is set to 300 seconds with a minimum history length of 60 epochs, which triggers activation of the power manager at the five hour mark in our benchmark traces.

Our results show agreement between a-priori simulation and a-posteriori observation. We repeat simulation and real world runs 12 times (a total of 120 hours) for each trace separately and compute the averages. For visualization, two exemplar runs from the benchmarks are shown in the graphs in Figure 1. The figures depict the activity of the power manager over time. The y-axis represents the number of cores used, normalized to maximum capacity. The x-axis represents time in one hour (3600 seconds) intervals. The solid line shows the number of cores occupied by instances in the cluster while the dotted line shows the number of cores available on awake nodes. The activation of the power manager can clearly be seen at the five hour mark. With changes in utilization, a fluctuation of the number of awake nodes can be observed. Due to the high frequency of these changes in our registration traces, we expect the impact of inaccuracies in the simulation to be exacerbated.

Tables III and IV show average utilization and uptime (expressed as fractions, respectively) per node and their standard deviations using the synthetic exponential trace. The counterparts for the synthetic lognormal trace can be found in Tables V and VI. For this experiment, we are concerned with numerical accuracy and do not adjusted the power savings for the power manager's warmup period. We find a good match between simulation and real world observation, with the largest per-node difference of 2%.

Note that in our initial runs (omitted for brevity) the registration of both, core utilization and up time between simulated and measured exponential runs did not seem to match as precisely as we had anticipated. In particular, the utilization and uptime of nodes seemed to differ to a greater extent than

TABLE VII: Summary of Private Cloud Dataset Characteristics

Data Set	Nodes	Cores/Node	Time Period	Description
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
DS5, DS6	31	32	Nov. 2013 to Dec. 2013	Large company with 50,000 to 100,000 employees

we had hoped. Investigating the cause of this inconsistency, we discovered an implementation bug in the power manager that we integrated into Eucalyptus. This discrepancy illustrates an ancillary benefit to trustworthy simulation. By working with a perturbative model we were able to anticipate the degree of accuracy we could expect and thus launch a targeted debugging effort when we did not achieve it.

C. Power-aware scheduler at scale

The results described in the previous section show that the simulation of Eucalyptus with the power-optimizing scheduler match the observations of an actual Eucalyptus implementation of the scheduler to an error of less than 2% at scales that are feasible to test. In this section, we use the simulator to study the effects that the scheduler would have achieved in production settings had it been available and deployed.

To do so, we run the simulator using traces gathered from the logs generated by Eucalyptus when run in several production settings. The commercial enterprises who donated their Eucalyptus logs to the project asked not to be identified specifically. Table VII summarizes the node and core counts for each commercial trace, its duration, and a description of the size of the business. We number the datasets as DS2, DS3, DS5, and DS6 as they are part of a larger collection of data sets. The anonymized traces from the collection are available to the research community from [18].

We do not replay these traces through a "live" installation of Eucalyptus because each of these traces spans several months in real time. Further, in order to observe the power-optimizations from a working system, it would have been necessary to recreate the specific deployments that generated each trace. Note, however, from Table VII, that the core and node counts in these production deployments are modest. Production enterprises are often partitioned into smaller units both to enhance fault isolation and to allow resource expenses to better track business unit organization.

Thus at these scales, the power-optimization results are likely to be nearly as accurate as those shown in Subsection III-B. An inspection of the source code indicates that any additional overhead introduced by the additional nodes and cores would be covered by the perturbation terms in the model with one important caveat. At the time these traces were generated, the power-optimization scheduling algorithm did not exist nor had we begun its development. Thus we lack the specific hibernation and wake-on-lan response times that are necessary to parameterize the model. In the absence of this data, we use the empirical samples from our test cloud in its place. The result is an accurate simulation of what the efficacy would have been if the machine power-cycling performance response were the same as it is in our laboratory.

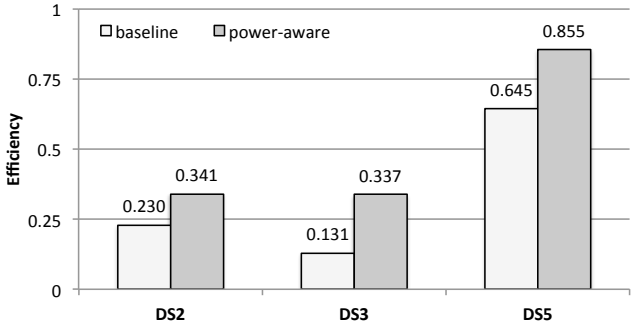

Fig. 2: Comparison showing power-efficiency of the power-aware scheduler with bars representing baseline (left) and power-aware (right) efficiency for three production traces (DS2, DS3, and DS5).

Of the three production traces, two are nine months in length with highly variable resource demand. The third trace is one month in length and has a very regular workload. Further, we set the power-manager epoch time to 1000 seconds, as suggested by the original authors, and use the same 95% responsiveness SLA as before. Also, the history length for samples takes by the power-optimizing scheduler is set to 2000 samples.

To predict efficacy, we define power efficiency to be the total CPU time for all nodes normalized to its theoretical maximum possible, divided by the uptime of active nodes normalized to its always-on baseline (as shown in Equation 1).

$$efficiency = \frac{total\_cpu\_time/max\_cpu\_time}{total\_uptime/max\_uptime} \quad (1)$$

Recall from Subsection III-B that CPU time is the amount of CPU time used by a node to run the VMs assigned to it and uptime is the total duration that a node is in the powered-up or waking state. As such, this formulation of efficiency captures the degree to which the power used by the system is used to run VMs.

We run a Monte-Carlo simulation 30 times for each trace, once with power manager enabled, once without. The results are shown in Figure 2. Error bars have been omitted due to the minimal deviation of the averages between runs. For the variable DS2 and DS3 the efficiency increases by a factor of 1.5 and 2.6 respectively, and for the constant DS5 by 1.3. While the relatively small improvement for the constant workload DS5 seems intuitive, the differences between DS2 and DS3 are not obvious at first. Close investigation shows that DS2 contains requests that demand access to the whole cluster after long periods of inactivity while DS3 has users demand

small batches several times before issuing a large request. With this difference, the power manager becomes more conservative in its predictions for DS2 compared to DS3, which results in lower overall power-savings and efficiency.

Note that in our initial runs of the long-term traces (omitted for the sake of brevity) we observed a large miss-percentage for the first two traces with our implementation of the power manager. The simulation and implementation agreed, but together they did not meet the SLA guarantees that the scheduling algorithm should have obtained. In communication with the authors of the algorithm, we found a discrepancy with our implementation when correcting for very long periods of inactivity on the cluster. We corrected our implementation, both in the simulator and for Eucalyptus itself, re-validated on our test bed and then executed the long-term traces again. This time the SLAs were met without exception. Due to faster-than-realtime simulation the turnaround time for debugging, updating and re-evaluation correspond to a fraction of the time required for real-time testing alone.

By replaying real-world traces, the simulator helps to determine the impact of subtle differences in the workload before the production deployment of a new scheduler. This is especially true when synthetic traces do not exhibit all the properties of production workloads. This makes testing efforts more robust and provides insights for planning the deployment of a new IaaS resource manager.

D. Capacity Planning

Aside from software development and testing, trustworthy simulation can inform capacity and business planning. So far, we have solved one-dimensional, monotonic problems for testing a power manager's efficiency under a single SLA constraint. In contrast, stakeholders in enterprises have to consider multidimensional problems with consideration given to capital and operating expenses, ease of use, robustness of a system and transition policies, among others. Using the simulator to test different platform configurations against a recorded trace, we find Monte-Carlo simulation provides additional insights to inform trade-offs between cost and expected quality of service and simplifies the decision-making process.

In this experiment, we use a 1-month section of a production trace (depicted in Figure 3). Given this workload and our use of the power manager, we investigate how many nodes we can remove from the cluster while still meeting our chosen SLAs (95% responsiveness and 99% start request acceptance). To enable this, we run the simulation with the power manager activated (PM) and without (base) and incrementally remove nodes from the base configuration until SLA violations occur. We use the same configuration of the power manager from previous experiment and register the system via the logged request delays of the real system. We conservatively assume 600 seconds for node power-up.

Figure 4 depicts the aggregate power-efficiency and request acceptance rate on the y-axis and the reduction in the number of nodes over the baseline system on the x-axis. We omit plotting misses due to wake-on-lan as the SLA is never violated in this experiment. There are two interesting insights revealed by the data. First, after a reduction by 8 nodes we cross the threshold of diminishing returns for the non-power-aware case,

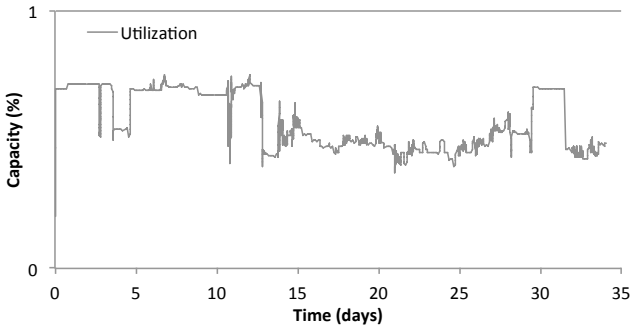

Fig. 3: This production trace of an over-provisioned cluster with a fixed workload is the foundation of the down-sizing scenario.

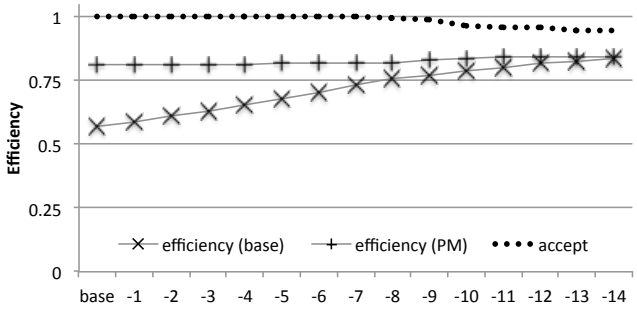

Fig. 4: Power-efficiency increases while request acceptance rate decreases as node count goes down in the base case. The power-aware scheduler guarantees constantly high efficiency.

while the maximal reduction lies at 9 nodes before violating the acceptance SLA. However, even guaranteeing a 99.9% acceptance SLA would still allow for a reduction by 6 nodes (about 20% of the cluster). Second, for the power-aware case, we notice that the efficiency is almost constant and independent of the node count for the cluster's specific workload.

With this data about the non-parametric power-manager in hand, decision-makers can focus their attention on other aspects of the capacity planning problem. Furthermore, if different levels of quality of service guarantees are being considered, reliable estimates about their expected cost can be obtained via simulation.

E. Capacity planning for scale-out workloads

Our final use-case is capacity planning for scale. Simulation gives decision-makers the ability to make reliable forward-looking estimates about the hardware requirements for an expected workload without acquiring or renting all necessary resources (*i.e.* servers and infrastructure) for testing ahead of time.

In this set of scale-planning experiments, we run the simulator as a "parameter sweep" varying both the number of nodes in the simulation, and the intensity of the workload (by changing the mean arrival time) independently. Thus each simulation depicts the behavior of the cloud at a given size for

	1.50	2.00	2.50	3.00	3.50	4.00
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	-	-	-	-	0.42
4	-	-	-	-	0.50	0.31
5	-	-	-	0.66	0.40	0.25
6	-	-	-	0.56	0.33	0.21
7	-	-	-	0.48	0.28	0.18
8	-	-	0.69	0.42	0.25	0.16
9	-	-	0.62	0.37	0.22	0.14
10	-	-	0.56	0.33	0.20	0.12
11	-	-	0.51	0.30	0.18	0.11
12	-	0.78	0.46	0.28	0.17	0.10
13	-	0.72	0.43	0.26	0.15	0.10
14	-	0.67	0.40	0.24	0.14	0.09
15	-	0.63	0.37	0.22	0.13	0.08
16	-	0.59	0.35	0.21	0.12	0.08
17	-	0.55	0.33	0.20	0.12	0.07
18	-	0.52	0.31	0.19	0.11	0.07
19	0.83	0.49	0.29	0.18	0.10	0.07
20	0.79	0.47	0.28	0.17	0.10	0.06

	1.50	2.00	2.50	3.00	3.50	4.00
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	-	-	-	-	0.42
4	-	-	-	-	0.51	0.39
5	-	-	-	0.67	0.48	0.38
6	-	-	-	0.59	0.47	0.38
7	-	-	-	0.58	0.47	0.37
8	-	-	0.71	0.57	0.46	0.37
9	-	-	0.67	0.57	0.46	0.36
10	-	-	0.66	0.56	0.45	0.36
11	-	-	0.66	0.56	0.45	0.35
12	-	0.79	0.66	0.56	0.44	0.35
13	-	0.75	0.65	0.55	0.44	0.35
14	-	0.74	0.65	0.55	0.44	0.34
15	-	0.73	0.65	0.55	0.43	0.34
16	-	0.73	0.64	0.54	0.43	0.34
17	-	0.72	0.64	0.54	0.43	0.33
18	-	0.72	0.64	0.54	0.42	0.33
19	0.84	0.72	0.64	0.53	0.42	0.32
20	0.80	0.72	0.63	0.53	0.41	0.32

Fig. 5: Parameter-sweep predicts changes in power-efficiency with increasing node count (y-axis) and workload intensity (Lognormal arrival, μ on x-axis, $\sigma = 1.0$) at the baseline (left) and with power-aware scheduler (right).

a given workload intensity. The simulator assumes a scale-out workload, *e.g.* 3-tier web applications or MapReduce jobs, and computes the expected power-efficiency for a given platform size.

We configure a virtual cluster of nodes with properties similar to the cluster used as our real-world test bed, but double their core capacity. The workload is generated from a lognormal distribution similar to the one we use in our registration experiments (*c.f.* Table I). We also set the SLA for not incurring a power-up delay to be 95% (and the scheduler achieves this probabilistic SLA in each case). In addition, because the number of nodes at some point in the parameter sweep may be insufficient to run the offered load (the cloud is out of resources), we only report results for the cases where at least 99% of the simulated VMs were able to run. The power-up delay for a node is again assumed to be 600 seconds.

Figure 5 shows the mean power efficiency (as computed in Equation 1) for each combination of intensity and node count. Node counts on the vertical dimension of the figure correspond to the number of nodes the cloud has configured. The intensity value (horizontal dimension) show the value of μ used in each lognormal parameterization ($\sigma = 1.0$ in each case). Parameter-combinations that fail to achieve the target SLAs are marked "-" in the figure. Each entry covers a simulated time-frame of 60 days. The left-hand table in Figure 5 shows power efficiency for different combinations of intensity and cloud size without the power-optimizing scheduler and the right-hand table shows the same with the scheduler activated. We use a heat map to color the efficiency numbers (green for high, red for low) for each combination.

For example, in the left-hand table, the entry for 19 nodes with $\mu = 1.5$ corresponds to a power efficiency of 0.83 (colored green). Thus a cloud with 19 nodes experiencing a workload with lognormal inter arrival times ($\mu = 1.5$, $sigma = 1.0$) and lognormal durations ($\mu = 3.8$, $\sigma = 1.0$ from Table I) for 60 days would achieve a power efficiency of 0.83 without the power-optimizing scheduler. The same entry in the right-hand table shows that with 19 nodes and $\mu = 1.5$ the power-optimizing scheduler achieves an efficiency of 0.84.

Unsurprisingly, these results indicate that as the inter arrival time goes down (smaller values of μ) the efficiencies converge to a high value both with and without the power-optimizing scheduler. These extreme cases correspond to the cloud being "full" almost all of the time leaving little efficiency to be gained by powering nodes on and off. At the other extreme, when inter arrival times are larger (large values of μ) the power-optimizing scheduler has more of an opportunity to save power. Indeed just looking at the heat map coloration of both tables shows the trend in efficiency in both dimensions. All green entries in the regular scheduler are green for the power optimizing scheduler (it does no harm). In addition, the power-optimizing is "greener" across all entries and never "red."

With faster-than-realtime simulation we can perform parameter sweeps across large ranges of configuration parameters, such as cluster size and workload intensity. Due to the embarrassingly-parallel nature of Monte-Carlo simulation, parameter-sweeps can be performed anywhere, from a personal laptop to a group of workstations, with a flexible trade-off between accuracy and wait time. Initial results are available within minutes, followed by increasing degrees of confidence and minimal convergence. The results for this experiment total at $450,000$ VM starts and $7,200$ days (about 20 years) of simulated time, and were generated on commodity laptop hardware within 8 hours. This demonstrates the practical ability of this approach to quickly estimate the impact of different workloads, additional hardware or new resource-allocation policies.

A core interest of cloud operators is the trade-off between risking service disruption due to increases in load or the introduction of new technologies, and unnecessary capital- and operation-expenses. In our example, the data reveals that the power manager can be used safely and without negative impacts on availability, independently of the platform size and workload intensity. At high levels of utilization its impact is marginalized, however, which can inform decisions based on the expected workload and cluster size.

Another insight that can be gained from the parameter-sweep is the amount of resources required to achieve a specific target utilization of the cluster (which equals cluster efficiency of the non-power-aware baseline). In our example, a mean utilization target of 50% demands 3 nodes for the light workload ($\mu = 4.0$) and moves up to 4, 7, 12, 19 and further with increasing workload intensity. Depending on the workload, the non-linear interactions between utilization and SLA constraints are hard to estimate analytically or with rules-of-thumb. Accurate simulation overcomes this limitation and helps allocate resources efficiently inside and around the cloud.

Monte-Carlo simulation offers a high degree of flexibility within its defined parameter space. Results can be generated quickly and refined iteratively. When registered to the properties of a production cluster, accurate predictions about efficiency and reliability become possible and help decision-makers make better trade-offs.

IV. Related Work

Empirical evaluation of distributed systems technologies has a long tradition in Computer Science. Recently, Gustedt

et al. [20] has classified methodologies and recommends best practices for performing experimental validation of large scale systems using real-scale experiments, emulation, benchmarking, and simulation. The authors discuss the importance of ab initio (high-level, imprecise, easily composable, and extensible simulation for use in comparative analysis and exploration) and validated simulation (simulation that produces behavior that matches that of a real system with low error). Grid research has spawned multiple simulators, including SimGrid [21] and GridSim [22]. The former provides validation for some simulation components, the latter is an ab initio approach. Such systems are challenging to use for cloud systems since they lack support for on-demand resource allocation, elasticity, and other cloud features.

To facilitate cloud research on a broad scale, a series of domain-specific simulators have been developed by the community. In particular, CloudSim [9] and NetworkCloudSim [12] allow simulation of large-scale clouds using an ab initio approach. CloudAnalyst [11] extends CloudSim to facilitate simulation of globally distributed applications such as social networks. These simulators model system components and workloads from the bottom up and compose them into large-scale configurations. Their approach is very flexible and extensible, but does not provide the accuracy guarantees necessary for evaluating production-quality cloud components. Alternatives allow for real-scale (in-situ) experimentation [23], [24], but their use is limited for practical reasons (e.g. time, available cluster size, budgetary constraints).

GreenCloud [10] is a simulator that focuses on exploring the energy consumption of different datacenter network architectures. It builds upon the NS2 [25] network simulator, and estimates the efficiency of hibernation and power-stepping strategies for servers and network components. Workloads and hardware are modeled with differing compute and communication capacities and, similar to our approach, the authors consider SLA requirements. GreenCloud inherits network level accuracy from NS2, but does not consider accuracy of per-node resource allocation or empirical validation of predictions.

EMUSIM [26] uses emulation of Bag-of-Task applications to extract performance properties and simulate their behavior at larger scale more accurately. An evaluation step ensures that emulation and simulation agree at observable scales. We similarly obtain empirical measurements at small scale and scale them up in simulation. In contrast, our approach focuses on cloud infrastructure components (not individual applications) and makes predictions about the utilization and resource-use of the cloud as a whole.

RC2Sim [27] is an integrated simulation and emulation environment for testing of production-quality cloud management code. It provides a compatible web API and emulates distributed operations, such as file transfers and remote shell access, on a single physical machine. This prior work focuses on functional testing of code rather than, as we do in this paper, on accurate simulation of resource-usage and execution time.

DCSim [13] simulates IaaS clouds with a specific focus on dynamic power- and SLA-optimization. The authors use tiered scale-out-type workloads and evaluate the advantage of VM migration and replication strategies over static provisioning. Similar to our work, they consider node power states and transitions, but do not perform an empirical evaluation of the simulation results.

GDCSim [14] addresses the thermal aspects of power-management in data centers by integrating existing models. Specifically, It investigates the interaction of workloads and resource management policies with heat dissipation and fluid dynamics of different physical data center layouts. Empirical validation of predictions is left for future work.

MDCSim [28] uses detailed models and hardware specifications to simulate the impact of networking infrastructure on web applications. The authors augment their simulation model with workload characteristics obtained from real-world measurements and make accurate, empirically validated, predictions about latencies for the RUBiS benchmark. The work is similar to ours in terms of allowing trustworthy capacity planning, but targets 3-tier web applications instead of generic IaaS cloud infrastructures.

Research in data center power-efficiency [29], [30], [31] predates the call for power-proportional computing [32], but has gained substantial traction and public interest since [33]. Recent work in datacenter power efficiency is surveyed by [34] and illustrates the significant potential of energy- and cost-savings via pro-active power-management. The power-aware scheduler that we implement in this paper, was originally proposed in [17]. It uses a quantile predictor to estimate the number of hot spares needed to maintain a configurable responsiveness SLA. We also gained insights about the specific workloads in private clouds (which we use to drive our synthetic workload generation) from our previous work described in [19].

V. Conclusion

Simulation plays a key role in performing experimental exploration into large scale systems. As such, simulation has significant potential for facilitating research and experimentation with cloud computing infrastructures. Cloud research is important for advancing the state of the art in cloud performance, scale, energy efficiency, and fault tolerance, among other features. However, the wide spread use and commercial viability of cloud computing requires that simulated results be sufficiently trustworthy (accurate) to ensure adoption in production settings and to justify the engineering effort required to achieve production levels of performance and reliability. Extant simulation systems typically trade off validation and accuracy for configurability and exploratory power, through the use of ab initio techniques that facilitate comparative evaluation of cloud components and application behavior.

In this work, we present a new methodology for facilitating trust in the simulation of cloud components through the use of a tool employed in the physical sciences for simulation called perturbation theory. Using this methodology, we derive a parsimonious model from a real cloud infrastructure (in our case a Eucalyptus private cloud) for the cloud component under study (in our case a scheduler). We then perturb the model using statistical sampling to represent unmodeled behavior to facilitate simulation speed and scaling. We incrementally add parameters (component inputs) to the model (incorporating key unmodeled behavior) until we achieve an acceptable level of accuracy for the component, relative to the real system. It

requires, however, that we have access to a production-quality cloud that we can interrogate and validate against.

We find that this perturbation approach achieves high accuracy for simulating an existing system and works well for evaluating the resource efficiency of cloud schedulers. As such, we use the derived simulation model to rapidly implement and evaluate a new power-aware cloud scheduler and investigate its performance in a number of scaled (larger size and faster than real time) experiments for capacity planning.

ACKNOWLEDGMENTS

This work was funded in part by NSF (CNS-0905237, CNS-1218808, and ACI-0751315) and NIH (1R01EB014877-01).

REFERENCES

[1] "Amazon Web Services home page," http://aws.amazon.com/.

[2] "Google Cloud Platform," [Online; accessed Aug-2014] "https://cloud.google.com/".

[3] "IBM SoftLayer," [Online; accessed Aug-2014] "http://www.softlayer.com/".

[4] "Rackspace Cloud," [Online; accessed Aug-2014] "http://www.rackspace.com/cloud/".

[5] 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.

[6] Eucalyptus Systems Inc. (2013, Jun.) http://www.eucalyptus.com. [Online]. Available: http://www.eucalyptus.com

[7] "OpenStack," [Online; accessed Aug-2014] "http://www.openstack.org/".

[8] "CloudStack," [Online; accessed Aug-2014] "http://cloudstack.apache.org/".

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. D. Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, no. 1, pp. 23–50, 2011.

[10] D. Kliazovich, P. Bouvry, and S. U. Khan, "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers," The Journal of Supercomputing, vol. 62, no. 3, pp. 1263–1283, 2012.

[11] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "Cloudanalyst: A CloudSim-based visual modeller for analysing cloud computing environments and applications," in Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. IEEE, 2010, pp. 446–452.

[12] S. K. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on. IEEE, 2011, pp. 105–113.

[13] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management," in Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm), Oct 2012, pp. 385–392.

[14] S. K. S. Gupta, R. Gilbert, A. Banerjee, Z. Abbasi, T. Mukherjee, and G. Varsamopoulos, "GDCSim: A tool for analyzing Green Data Center design and resource management techniques," in Green Computing Conference and Workshops (IGCC), 2011 International, July 2011, pp. 1–8.

[15] "Perturbation Theory," http://en.wikipedia.org/wiki/Perturbation_theory.

[16] "Monte Carlo Method," http://en.wikipedia.org/wiki/Monte_Carlo_method.

[17] R. Wolski and J. Brevik, "QPRED: Using Quantile Predictions to Improve Power Usage for Private Clouds," Computer Science Department of the University of California, Santa Barbara, Santa Barbara, CA 93106, Tech. Rep. UCSB-CS-2014-06, September 2014.

[18] ——. (2013) http://www.cs.ucsb.edu/~rich/workload. [Online]. Available: http://www.cs.ucsb.edu/~rich/workload

[19] ——, "Using Parametric Models to Represent Private Cloud Workloads," Computer Science Department of the University of California, Santa Barbara, Santa Barbara, CA 93106, Tech. Rep. UCSB-CS-2013-05 (to appear in IEEE Transactions on Services Computing), October 2013.

[20] J. Gustedt, E. Jeannot, and M. Quinson, "Experimental Validation in Large-Scale Systems: a Survey of Methodologies," Parallel Processing Letters, vol. 19, no. 3, pp. 399–418, 2009.

[21] H. Casanova, "Simgrid: A toolkit for the simulation of application scheduling," in Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on. IEEE, 2001, pp. 430–437.

[22] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," Concurrency and computation: practice and experience, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[23] A. I. Avetisyan, R. Campbell, I. Gupta, M. T. Heath, S. Y. Ko, G. R. Ganger, M. A. Kozuch, D. O'Hallaron, M. Kunze, T. T. Kwan et al., "Open Cirrus: A global cloud computing testbed," Computer, vol. 43, no. 4, pp. 35–43, 2010.

[24] M. Silva, M. Hines, D. Gallo, L. Qi, R. K. Dong, and D. D. Silva, "CloudBench: Experiment Automation for Cloud Environments," in Cloud Engineering (IC2E), 2013 IEEE International Conference on, March 2013, pp. 302–311.

[25] "NS-2 Network Simulator," http://www.isi.edu/nsnam/ns/.

[26] R. N. Calheiros, M. A. Netto, C. A. D. Rose, and R. Buyya, "EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications," Software: Practice and Experience, vol. 43, no. 5, pp. 595–612, 2013.

[27] D. Citron and A. Zlotnick, "Testing large-scale cloud management," IBM Journal of Research and Development, vol. 55, no. 6, pp. 6–1, 2011.

[28] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, "MDCSim: A multi-tier data center simulation, platform," in Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on. IEEE, 2009, pp. 1–9.

[29] 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, vol. 180. Barcelona, Spain, 2001, pp. 182–195.

[30] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in ACM SIGOPS Operating Systems Review, vol. 35, no. 5. ACM, 2001, pp. 103–116.

[31] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," ACM SIGARCH Computer Architecture News, vol. 35, no. 2, pp. 13–23, 2007.

[32] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," Computer, vol. 40, no. 12, pp. 33–37, 2007.

[33] R. Brown et al., "Report to congress on server and data center energy efficiency: Public law 109-431," 2008.

[34] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," Future Generation Computer Systems, vol. 28, no. 5, pp. 755–768, 2012.