Stochastic Queuing Simulation for Data Center Workloads

David Meisner
meisner@umich.edu

Thomas F. Wenisch
twenisch@umich.edu

Advanced Computer Architecture Lab
The University of Michigan

Abstract

Data center systems and workloads are increasing in importance, yet there are few methods for evaluating potential changes to these systems. We introduce a new methodology for exascale evaluation, called Statistical Queuing Simulation (SQS). *At its heart, SQS is a parallel, large-scale stochastic discrete time simulation of generalized queueing models that are driven by empirically-observed arrival and service distributions. SQS provides numerous practical advantages over alternative large-scale simulation techniques (e.g., trace-driven simulation), including statistical rigor and reduced turnaround time. We detail our methodology, workload suite, and practical concerns associated with them. To demonstrate our technique, we carry out a case-study of data center power capping for 1000 servers. Finally, we discuss open research challenges for making SQS more robust.*

1. Introduction

Data center workloads are increasing in importance. With time, more computation is moving into *Warehouse-Scale Computers* (WSCs, or, "The Cloud"). Despite this trend, the systems community is only beginning to understand these workloads and their implications. Furthermore, because of their infancy, there are few, if any, established methods for monitoring, designing, and evaluating these systems.

Evaluating a proposed exascale system is challenging. The research community has achieved many successes in engineering and benchmarking individual servers. However, the challenge of exascale research lies in scaling to many thousands of servers. While prototyping or implementing novel systems might be most appealing, the cost and scale of data centers makes such deployments impractical. Instead, modeling and simulation are both promising avenues for quantitative evaluation at scale. The research literature includes a wealth of theoretical and statistical modeling methods, however these analytic solutions often fail to handle the complexity of real systems. Simulation offers greater flexibility and fidelity than analytic models, but care must be taken to maintain statistical accuracy and limit simulation time.

Our evaluation solution is a hybrid of implementation, benchmarking, modeling and simulation that we call *Stochastic Queuing Simulation* (SQS). SQS is based on a stochastic discrete-time simulation of a generalized system of queueing models driven by empirical profiles of a target workload. We present a case for our methodology, detail its theoretic foundations and provide our workload suite used to drive it. Finally, we demonstrate the utility of SQS through a case study of cluster level power capping.

1.1 Simulation and Modeling Challenges

Where possible, the use of closed-form analytic equations for system performance is preferable; these equations lend insight into the nature of the systems and require no simulation time. Most analytic solutions available in the queuing theory community rely on independence assumptions and Poisson processes [11]. Unfortunately, many important server workloads exhibit bursty behavior and/or long tail distributions which break these assumptions [10]. The most general and applicable queuing model for which numerous performance measures have been derived is the M/G/1 queue. This model accounts for arbitrary job lengths, but is not general enough to model bursty arrivals, many-core servers or novel power management modes.

Ideally, one would like to model real workloads from observations on actual systems (i.e., using empirical characterizations of arrival and service processes). To model arrival or service processes with arbitrary distributions (i.e. 'G' in queuing notation for 'general') one must analyze a G/G/1 or G/G/k queue (General interarrival and service distributions and 1 or k servers). Unfortunately, such models have to-date remained analytically intractable within the queuing theory community [7, 8]. To handle complex systems, analytic models must relax assumptions and make approximations to conform to analytically-tractable queues; the more of these relaxation that are made, the further from the real system the model becomes.

In contrast, trace driven simulation, where a sequence of events captured on a real system is replayed on a model of an alternative system, allows for analysis of arbitrarily

complex systems with good fidelity. Traces are valuable because they represent the exact behavior of the target system. Without statistical reasoning however, it is unclear how large these traces must be, or what fraction must be replayed during simulation. Moreover, when captured over thousands of servers, traces can become unwieldy (gigabytes to terabytes in size) and require long simulation runs. Moreover, most trace-driven simulations are inherently serial.

1.2 Advantages of Stochastic Queuing Simulation

Compared to trace-driven simulation or analytic modeling, SQS provides a number of advantages.

Statistical rigor. SQS uses statistics to give probabilistic guarantees for its outputs. We build upon the large collection of methods in sampling and statistics. By rigorously defining convergence criteria for our simulations, we save on simulation time while achieving desired levels of prediction accuracy.

Compact representation. Compared with methods such as trace-drive simulation, the data needed to characterize a workload for SQS is small. Collecting fine-grain traces at sub-second scale (which is needed for many Internet services [13]) requires significant storage for observations over long time periods. We have collected over 60GB of trace data; using such large data sets is burdensome. Instead, we can represent the same set of workloads with SQS distributions in well under 1MB.

Free of confidential information. Concerns about privacy and competitive advantage have made it difficult for the research community to gain access to a set of workloads that represent data center services. The inner workings of data center operations are tightly guarded trade secrets. It is no surprise that data, such as workload traces, have not been forthcoming from industry. Business and legal ramifications may prevent such data sets from ever being released [1]. In contrast, SQS needs only interarrival and service time distributions and power-performance models, which reveal only a specific and limited set of information about a particular service.

2. Stochastic Queuing Simulation

Stochastic Queuing Simulation (SQS) is a methodology for characterizing and simulating large-scale workloads (e.g., to evaluate new server configurations, scheduling policies, etc.). The technique builds upon analytic foundations, but adds simulation to account for data center workload properties that make closed-form solutions intractable. While pieces of these methods may be well known to queuing theorists or statisticians, they have not been presented in a cohesive manner, or widely adopted by the systems community.

2.1 Simulation

SQS is based on discrete-event simulation. This technique is well understood and we leverage the wealth of statistical techniques for computer system simulations [20]. Because of the relative simplicity of our simulator (e.g., compared to timing accurate microarchitecure simulators), we are able to gather large samples with reasonable simulation time. The high speed of discrete-event simulation is fortuitous; the large degree of variability in the performance metrics used to assess large-scale systems require large samples to yield estimates with high confidence.

2.2 Queuing Models

We briefly review the important highlights of queuing models. Detailed tutorials on queuing models may be found in [11]. A queuing model is composed of a collection of "servers" which process jobs. We model each data center server as a single queuing system; this queuing system may have multiple "servers" which correspond to individual cores in a multicore processor. Jobs arrive into the system according to an *interarrival time distribution* and their size (measured in time) is distributed according to a *service time distribution*. A *queuing discipline* must be chosen to determine how queued jobs are scheduled and processed; we limit our study to the *first in first out* (FIFO) discipline.

There are two reasons why designers must resort to queuing models instead of simpler per-job power-performance models. First, increasing the latency of a single job can have compounding effects on other jobs' latency. A job's average response time, $E[R]$, is dependent on its expected service time, $E[S]$, which represents the relative size of the job, and *also* on the time spent in a queue waiting for other jobs to finish, $E[T_Q]$. We can express this as:

$$E[R] = E[T_Q] + E[S] \tag{1}$$

Accordingly, simply understanding how job length is changed under various configurations is not sufficient; we must also understand how jobs interact. A salient example is a bursty workload where many jobs arrive at once, possibly incurring large queuing delays.

The second reason queuing models are necessary is to understand the timing of how concurrent jobs overlap. Modeling resource constraints of multiple simultaneous jobs (e.g., power, memory bandwidth, etc.) is critical to understanding how to design systems with concurrency. Many properties, such as full-system idleness, require observing the interaction of multiple jobs on the same system [13].

We can characterize a workload for queueing analysis by capturing its interarrival time distribution and service time distribution.

Interarrival Time Distribution. The interarrival distribution reflects the probabilities that job arrivals are separated

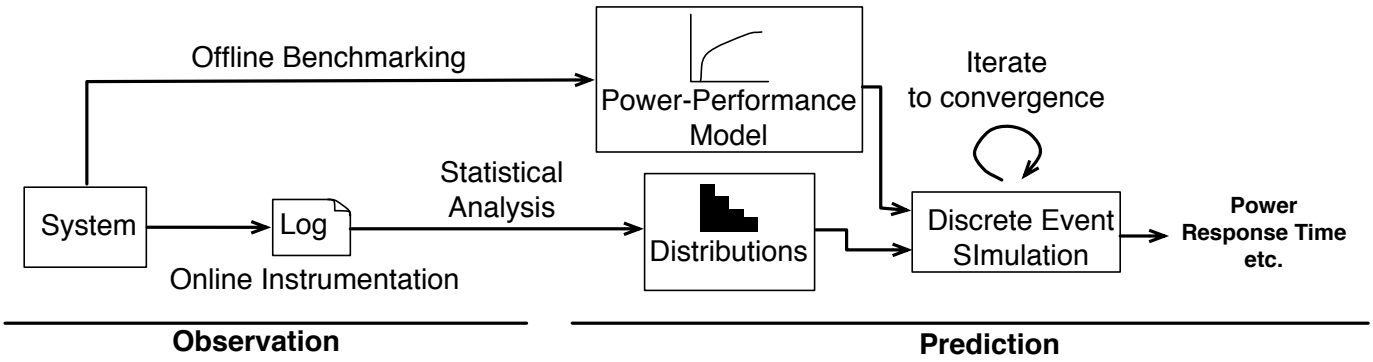

Figure 1: Stochastic Queuing Simulation

by given intervals. The rate parameter λ, provides the effective throughput of the system (job/s) and $1/\lambda$ is the average interarrival time.

Service Time Distribution. The service time distribution describes the probability of jobs of given sizes arriving at a server. μ is the rate parameter (job/s) for a server and $1/\mu$ is its average service time. Note that in statistics, μ typically denotes the average of a distribution, however, in queuing systems, it is $1/\mu$.

Coefficient of Variation. A useful statistic to characterize a distribution is its *coefficient of variation*, C_v.

$$C_v = \frac{\sigma}{1/\mu} \tag{2}$$

Where σ is the distribution's standard deviation. Unlike variance, this is a normalized statistic that can be compared across distributions. The service time distribution of commercial server applications tend to have high variance [10]. Furthermore, C_v roughly describes the burstiness of a distribution relative to an exponential (memoryless) distribution, which has a $C_v = 1$. Typical values of C_v for server applications range from one to 20.

2.3 Instrumentation

To characterize a workload's arrival and service distributions, a running server must be instrumented. The most straightforward method for collecting these distributions is to create an application-level log that records a timestamp of every job arrival and departure. This data can then be post-processed to find interarrival and service times.

Handling imperfect information. Often, it is infeasible to instrument an application's code (e.g., if the application is closed source or if a production system cannot be taken down for modification). We present two approximation techniques which allow researchers to infer interarrival and service times without application-level instrumentation. Though these approximations are imperfect, they nonetheless allow SQS to enable reasoning about performance changes for workloads that cannot otherwise be analyzed.

M/G/1 inference. If a service is user-facing (i.e., users interact with a server directly), with a relatively large user population, one can reasonably assume that arrival process is Poisson [11]. For a multi-processor server – M/G/k in queuing notation – we can leverage the Poisson assumption to break the system into k M/G/1 queues using *Poisson splitting* [11]. In this case, we can use parametric inference to fit the exponential distribution and use non-parametric inference to determine the service time distribution from idle and busy periods [14].

Light Utilization. Another approximation is possible for lightly utilized services – the most common case for data center applications. Under light utilization, queuing is probabilistically unlikely. So as $U \to 0$,

$$E[R] \approx E[S] \tag{3}$$

Furthermore, if there are a large number of servers and aggregate utilization is low, the probability that a job arrives and no server is available is low. So as $k \to \infty$, again,

$$E[R] \approx E[S] \tag{4}$$

Since the servers we monitored had 1-4 cores and low utilization ($\approx 20\%$), we use the latter technique to approximate their interarrival and service time distribution from their idle and busy patterns.

2.4 Power-Performance Models

One particularly important application of SQS is evaluation of various low-power modes. Idle low-power modes for CPUs are easily modeled in queuing systems; the transition penalties simply are added before jobs can begin/continue [13]. However, active low-power modes and non-CPU low-power modes require more complex models. For example, if a low-power mode degraded memory system response time,

the performance impact might vary across workloads. Predicting how much a given power-performance setting slows a given workload remains an open research challenge. Prior work [16] discusses modeling techniques for the physical infrastructure (power and cooling systems) in data centers.

3. Methodology

The flow of SQS is outlined in Figure 1. Two classes of models must be constructed for each server and workload to be studied: power-performance models and interarrival/service distributions. Power-performance models are constructed for various power mode or configuration tradeoffs through offline experiments (e.g., by measuring mean throughput and power demand under each power mode). Interarrival and service times must be collected from live systems (the real-world user behavior is a key aspect of the arrival distribution, making it critical to collect data from production environments where possible). This data collection can be quite challenging; the instrumentation must affect of the workload's performance minimally. Once these models are in place, various metrics can be extracted through queuing-based discrete even simulation.

3.1 Outputs

Extracting various performance metrics from SQS requires a number of statistical techniques to insure accuracy. These outputs may be derived from any performance metric captured in the discrete event simulator.

Simulations continue until target performance metrics meet a pre-specified statistical confidence, which we call simulation *convergence*. We describe a step-by-step process for achieving convergence. Simulations can output multiple metrics, but simulation length may increase as a result, as all metrics must reach convergence before the simulation completes.

Before detailing our simulation procedure, we recap the fundamental sampling mechanisms for mean and quantile performance estimates underlying our convergence criteria.

Means. To determine the confidence interval of average values (e.g., mean response time), we leverage standard techniques for large-sample analysis. According to the central limit theorem, the sampling distribution of a mean value estimate tends towards the normal distribution as sample size increases. Hence, we can determine the sample size needed for a given confidence by:

$$n_m = \frac{Z^2_{1-\alpha/2} \cdot \sigma^2}{\epsilon^2} \tag{5}$$

Where $Z_{1-\alpha}$ comes from the standard normal: it is the value of the standard normal distribution at the $(1-\alpha/2)^{\text{th}}$ quantile and is 1.96 for 95% confidence. σ is the sample standard deviation and ϵ is the half-width of the confidence interval.

Quantiles. Confidence intervals for quantile (e.g., the 95th-percentile latency) can also be derived using the central limit theorem [4].

$$n_q = \frac{Z^2_{1-\alpha/2} \cdot p(1-p)}{\epsilon^2} \tag{6}$$

The variables are the same as for mean estimates with the addition of p as the desired quantile. To find an exact quantile, one would need to record and sort all observations in the sample. However, space-efficient approximations using online algorithms are described in [3, 4].

Convergence. We consider a simulation to have converged when all target metrics have reached their predetermined confidence intervals. From these confidence intervals, we can derive statistically rigorous probabilistic claims regarding mean and quantile estimates. The interval ϵ gives the bounds of a confidence interval in terms of the target-metric's units (e.g., response time with $\pm$5ms). We normalize these value by the mean estimate such that we can report mean confidence across multiple output variables.

$$E = \epsilon / \bar{X} \tag{7}$$

The steps to simulate to convergence are:

1. **Warm to Steady State** - A simulation begins in an initial transient phase, where observations are biased by the initial simulation state (e.g., assuming that all queues are empty) and cannot be used. To avoid this cold-start effect, the simulation must be exercised for n_w observations until it reaches a steady-state where the performance metrics of a random observation are uncorrelated to the initial state. Unfortunately, a method for determining n_w has been the subject of years of debate [15]. To date, no rigorous method for automatically detecting steady-state is available. We conservatively choose large values for n_w (much larger than any busy interval we have observed in simulation).
2. **Find Lag Spacing** - Using successive observations from a queuing system simulation introduces bias into sample estimates because observations tend to be autocorrelated (i.e., nearby observations are not independent).

   However, it has been shown that if observations are sufficiently spaced apart, they are effectively independent. Determining this minimum spacing, l, is accomplished with the *runs-up* test detailed in [4]. The major consequence of this approach is that simulation length is inflated by a factor of l. Though a sample size of $N = n$ observations may be sufficient to achieve a given confidence in an i.i.d. draw, since $l - 1$ observations are discarded for every l taken, a total of $N = ln$ events must be simulated to achieve the target sample size. Further-

more, it has been shown that this method can increase the sampling variability, further increasing n [5].

3. **Choose Acceptable Confidence** - For each output performance metric, an acceptable$E$ must be chosen. For out simulations, we use $E = .05$ with a confidence level of 95%. This confidence interval can be interpreted to mean that, in 95% of simulation experiments, the true value of a performance metric will be within 5% of the reported estimate.
4. **Collect Sufficient Observations** - Finally, the simulation proceeds according to conventional methods for discrete event simulation. Each event generates a new observation, is fed to the runs-up test, and the simulation continues until a sufficiently-large sample has been observed to achieve convergence for all output variables.

4. Workloads

We present data that we have collected on the University of Michigan campus to drive SQS simulations. We include three distinct workloads: web mail (Mail), interactive login (Shell), and an Apache based web server (Web). In the next section, we will utilize these traces to evaluate simulation of the effect of power capping on a hypothetical cluster.

Figures 2-13 present empirical Probability Density Functions (PDFs) and Cumulative Distribution Functions (CDFs) of six instrumented production servers. Note that the left (for PDFs) and right (for CDFs) vertical scales differ drastically for a given workload and the horizontal scales vary greatly across workloads. The presented data demonstrate how it is useful to visualize both the PDF and CDF of each distribution.

Mail. We monitored a departmental mail server to produce the Mail workload. It exhibits slightly higher-than-exponential variation in its service distribution ($C_v = 3.6$). Most jobs for this system are short; typically email clients download a few emails from the server at a time.

Shell. The Shell workload monitors an interactive Linux login server, which are generally used to access files and run various jobs. Because users can run arbitrary software on these systems, the service distribution has a high C_v of 15 (although most of the density is below 20 ms indicating these jobs are generally short). Furthermore, the interarrival distribution does not follow the shape of an exponential. We conjecture that relatively few users access a single machine; arrival processes tend toward Poisson only under the aggregate behavior of a large population of users.

Web. A departmental web server's behavior is depicted in the Web workload. Both the interarrival and service distribution are fairly smooth; the service distribution has more variation than the arrivals. Because a large number of users access the web server, its interarrival distribution is quite close to exponential (C_v = 2), while the size of each web request has more variance ($C_v = 3.4$).

5. Case Study: Power Capping

We demonstrate SQS as an exascale evaluation method with a case study of *power capping* for a 1000 server cluster. Power capping is a data center power provisioning technique which enforces maximum power budgets over individual servers, racks and clusters [6, 12]. By making the observation that correlated power peaks are uncommon (i.e. the peak of the sum of power draws for servers is typically much less than the sum of maximum power draws of servers), power capping allows for under-provisioning of power delivery infrastructure by imposing hard limits on server power. We demonstrate the utility of SQS by simulating a hypothetical data center power capping deployment. From this simulation, we extract the level of power capping required and latency effect for our workloads.

Our case study uses a relatively simple power capping configuration; we wish to demonstrate the utility of our methodology rather than a sophisticated power capping strategy. Servers are assigned a *power budget*, the maximum power they may draw over a given interval. We use a fair, proportional budgeting mechanism such that every server gets a budget in proportion to its current utilization at each budgeting inteval. Budgets are calculated every 30 seconds. We use idealized *dynamic voltage and frequency scaling* (DVFS) as the power-performance throttling mechanism.

5.1 DVFS Power-Performance

To simulate power capping, we require a baseline server power model and a model for power savings and performance loss under DVFS. We use the linear model validated by [6] and [18]:

$$P_{\text{Total}} = P_{\text{Dynamic}} \cdot U + P_{\text{Idle}} \tag{8}$$

Where U is the average server utilization, P_{Dynamic} represents the dynamic range of the server's power, and P_{Idle} the idle power. We assume $P_{\text{Dynamic}} = 100$ W and $P_{\text{Idle}} = 150$ W (the maximum server power is 250W). For simplicity, we assume that the CPU is the only component with a dynamic power range such that:

$$P_{\text{Dynamic}} = P_{\text{CPU}_{\text{Dynamic}}} \left(\frac{f}{f_{\text{Max}}} \right)^3 \tag{9}$$

Where f is the operating clock frequency of the CPU. We assume that this frequency can be continuously scaled from $f = 1.0$ to $f = 0.4$, even though in practice these setting are discrete.

Next, we require a performance model to understand the slowdown imposed by various DVFS settings. The slow-

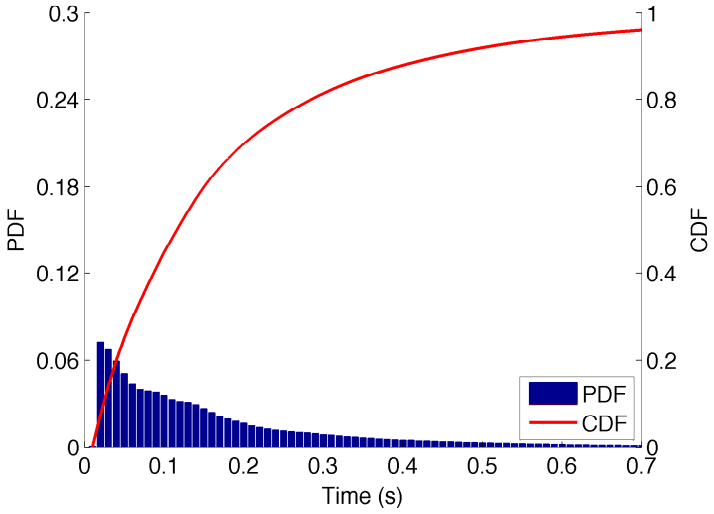

Figure 2: Mail Interarrival Distribution

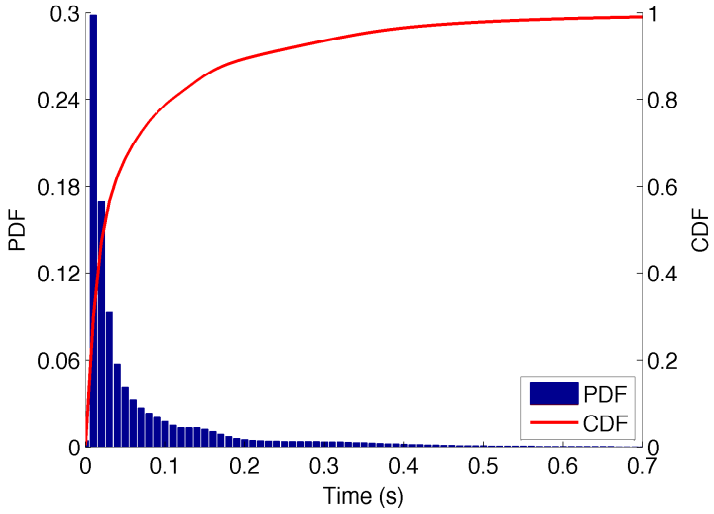

Figure 3: Mail Service Distribution

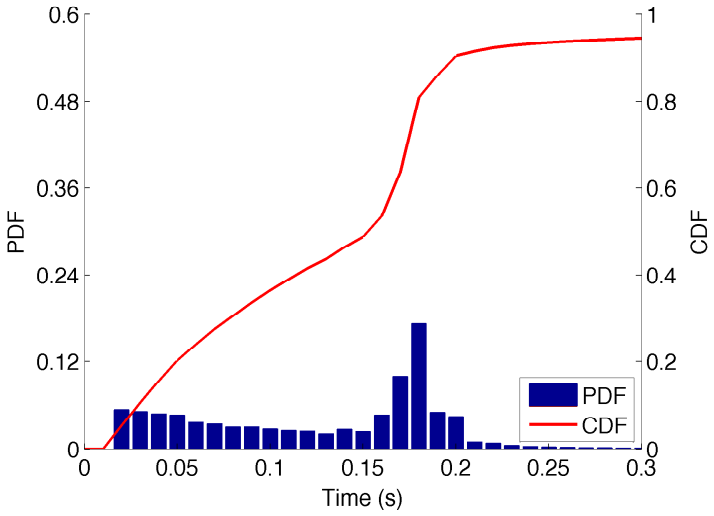

Figure 4: Shell Interarrival Distribution

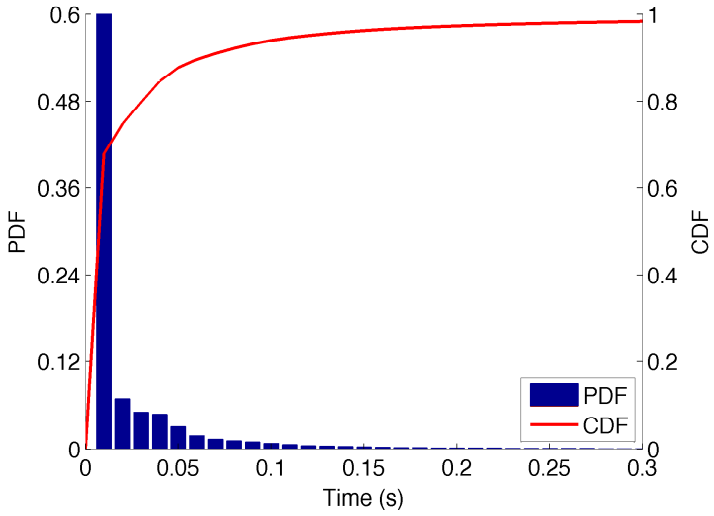

Figure 5: Shell Service Distribution

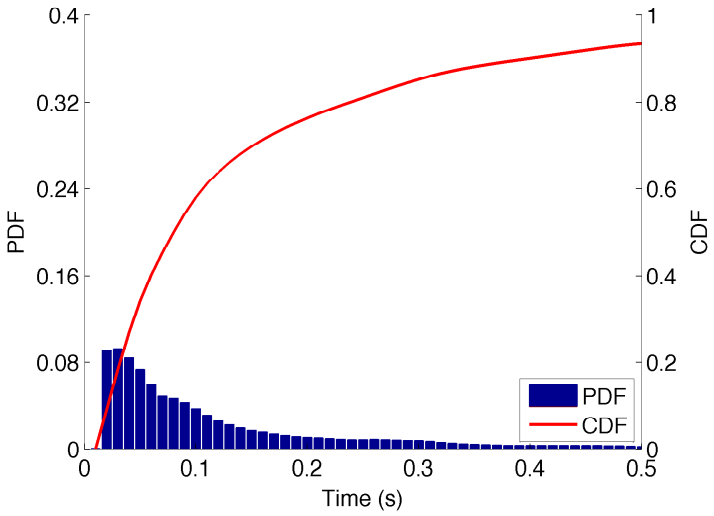

Figure 6: Web Interarrival Distribution

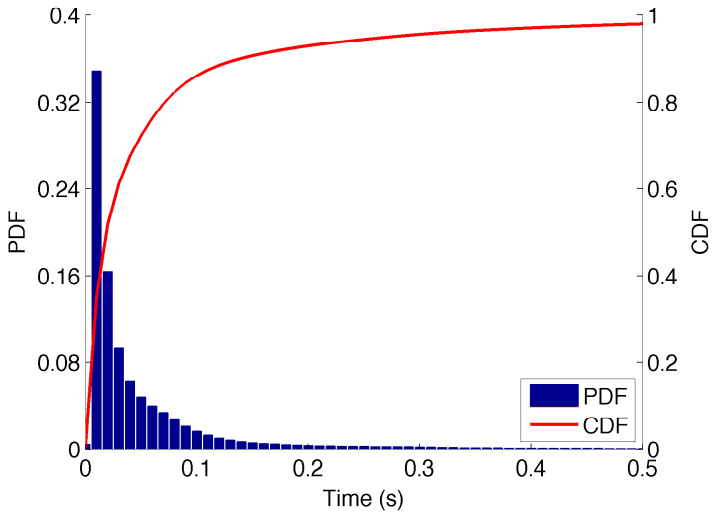

Figure 7: Web Service Distribution

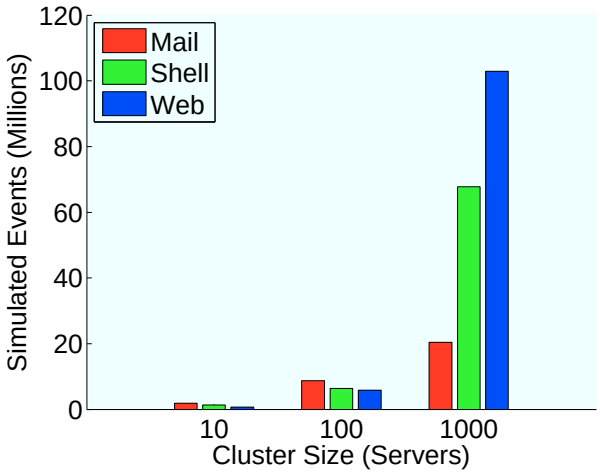

Figure 8: Simulation length depends on cluster size.

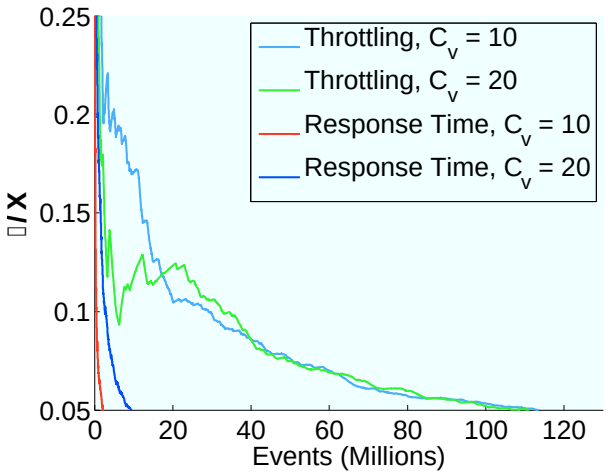

Figure 9: Simulation length depends on output variables.

down in service rate due to DVFS can be modeled as:

$$\mu' = \mu \cdot \alpha \cdot \left(\frac{f}{f_{\text{Max}}}\right) + \mu \cdot (1 - \alpha) \qquad (10)$$

For some α, which represents how "CPU-bound" an application is. We assume an α of 0.9, which would be typical of a CPU-bound application (e.g., LINPACK).

The power model given here is a simple example of the kind of model that can be used with SQS; the particular details of this model are not critical to the simulation approach.

5.2 Evaluation

We now show how the number events that must be simulated is affected by the size of the cluster, the output metric and the variability of the underlying distributions. Note that, because of the run-up test to avoid autocorrelation, each observation added to our sample requires the simulation of numerous events in the discrete-event simulation as previously described. The number of events is roughly proportional to simulation time and gives an implementation-independent measure of simulation performance.

Figure 8 shows how many events are simulated to achieve convergence for each workload across several cluster sizes. We find that increasing the cluster size increases the number of events *sub-linearly*. In other words, simulating a cluster of 1000 servers instead of 100 does not increase the number of events by 10x. This property is fortuitous and bodes well for scaling to larger simulations. It is also apparent that the properties of the individual workloads can substantially impact the simulation length, due to varying sample size requirements and lag spacing. For example, it takes nearly five times as many events to simulate web than mail for a 1000 server cluster.

We demonstrate how simulations approach convergence in Figure 9 in terms of the convergence quantity $\epsilon/\bar{X}$. A simulation will continue to run until all output metrics converge; for simplicity we only show the slowest for each configuration.

To illustrate the effect of variability on convergence time, we use synthetic workload distributions instead of our observed workloads for interarrival and service times. Using the gamma distribution, we are able to keep the mean value of a distribution constant, while changing C_v. First, we demonstrate the convergence of response time in a homogeneous cluster without power capping for two different values of C_v for the service time distribution. Clearly, C_v greatly impacts convergence; $C_v = 20$ requires over 8 times more events than $C_v = 10$. In wall-clock time, overall simulation length is relatively short; these simulations can be completed on the order of a minute.

Adding modeling of power capping ("Throttling") to the simulation substantially increases simulation length. In these simulations, the output metric is the average difference between requested and allocated power budget over all servers. This disparity in simulation length arises largely due to the long time scale over which power budgets are assigned (thirty seconds). Thousands of events must be simulated per observation to sample power budgets. Because the time scale of power capping is the limiting factor in simulation length, the variability of the service time distribution has little effect.

6. Future Challenges

We briefly describe a number of challenges for SQS we would like to address in future work.

6.1 Scaling Up

Unlike microarchitecure simulators, there are many opportunities for parallelization with SQS for low-complexity configurations. We briefly discuss various levels of parallelism

available and the challenges faced with additional complexity.

Server-Level Simulations. Statistical simulation of a single server provides ample parallelism. Each unique simulation can be considered a sample path. While a single sample path is not easily made parallel, we can run multiple simulations with different random seeds to achieve statistical convergence faster than along a single path. Each simulation may require a period of "warm-up", in which the system reaches steady state. As long as the warm-up time is significantly less costly than the rest of a sample path simulation, this technique will work well (alternatively these warm-up phases may be memoized and reused [19]). Furthermore, in practice we have found that to be useful a large parameter space must be simulated (e.g., how does the mean response time change with various DVFS settings and different workloads). We find that exploring parameter spaces alone can exhaust CPU resources.

Cluster- and Rack-Level Simulations. An important goal is to be able to provide accurate predictions for whole racks or clusters of servers. Making general statements about the complexity of such simulations is surprisingly non-trivial and dependant on the configuration. For example, if one wishes to evaluate the performance of web server front-ends, it is likely sufficient to simulate a single box in detail and safely assume that the results will hold across a rack with a good load balancer. On the other hand, simulation of a power capping technique for an entire cluster may require thousands of servers to be simulated with fine coordination. If one only wishes to determine the level of capping per server, all servers may be simulated independently and their power draws recorded.The capping levels may then be determined in a final reduce phase. However, if the servers need to adjust their power-performance state in response to other server, for example, the simulation may no longer parallel and servers must communicate at each time step. Therefore, developing a rigorous methodology for determining independence assumption is an important step towards simulating large systems and we leave this technique to future work.

6.2 Server-Independent Representation

A current limitation of SQS is that service time distributions are valid *only* for the machine on which they are measured. In other words, they represent the amount of time the observed server will spend on a job, rather than an independent notion of "work" to complete the job. This notion of job size limits our ability to observe service distributions on one system and report results for another. For example, one would not be able to accurately predict the service distribution when modifying an existing enterprise-class server to replace its processor with an energy-efficient, yet slower, alternative. Transforming an observed distribution from one server to another is non-trivial and we pose this transformation as an open research problem.

6.3 Non-Stationary Distributions

Many data center workoads show time varying utilization. Most services are heavily utilized during the afternoon, and show a significant decrease in traffic at night. One simple way to model this behavior is to modulate the mean of the interarrival distribution with time. However, a workload characterization will be necessary to insure that the shape of the distribution does not change with time.

7. Conclusion

We have presented Stochastic Queuing Simulation, our methodology for evaluation of data center systems. This technique enables researchers to rapidly evaluate the power-performance tradeoffs of new designs with statistical bounds. Our preliminary investigations of SQS scaling behavior suggest that it can scale to evaluate larger systems as we move forward into the exascale era.

References

[1] M. Barbaro and T. Zeller, "A Face Is Exposed for AOL Searcher No. 4417749," *New York Times*, 2006.

[2] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[3] E. J. Chen and W. D. Kelton, "Simulation-based estimation of quantiles," *WSC: Conference on Winter simulation*, 1999.

[4] Chen, E. Jack and Kelton, W. David, "Quantile and histogram estimation," *WSC: Conference on Winter simulation*, 2001.

[5] R. W. Conway, "Some Tactical Problems in Digital Simulation," *MANAGEMENT SCIENCE*, vol. 10, no. 1, pp. 47–61, 1963.

[6] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA: International Symposium on Computer Architecture*, 2007.

[7] A. Gandhi and M. Harchol-Balter, "M/G/k with Exponential Setup," Carnegie Mellon University, Tech. Rep. CMU-CS-09-166, September 2009.

[8] V. Gupta, M. Harchol-Balter, J. Dai, and B. Zwart, "On the inapproximability of M/G/K: whytwomoments of job size distribution arenotenough," *Queueing Systems*, 2009.

[9] M. Harchol-Balter, "Sample paths, convergence, and averages," 2005. [Online]. Available: http://courseweb.sp.cs.cmu.edu/ cs849/notes/

[10] M. Harchol-Balter and A. B. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *ACM Trans. Comput. Syst.*, vol. 15, no. 3, pp. 253–285, 1997.

[11] R. K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991.

[12] C. Lefurgy, X. Wang, and M. Ware, "Power capping: a prelude to power shifting," *Cluster Computing*, vol. 11, no. 2, pp. 183–195, 2008.

[13] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: Eliminating server idle power," in *ASPLOS: Architectural support for programming languages and operating systems*, 2009.

[14] N.H. Bingham and Susan M. Pitts, "Nonparametric inference from m/g/1 busy periods," *Stochastic Models*, vol. 15, no. 2, pp. 247–272, 1999.

[15] K. Pawlikowski, "Steady-state simulation of queueing processes: survey of problems and solutions," *ACM Computing Surveys*, vol. 22, no. 2, pp. 123–170, 1990.

[16] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and absracting total data center power," in *WEED: Workshop on Energy Efficienct Design*, 2009.

[17] S. Pelley, D. Meisner, P. Zandevakili, T. F. Wenisch, and J. Underwood, "Power routing: Dynamic power provisioning in the data center," *ASPLOS: Architectural support for programming languages and operating systems*, 2010.

[18] S. Rivoire, P. Ranganathan, and C. Kozyrakis, "A comparison of high-level full-system power models." *HotPower*, 2008.

[19] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe, "Simulation sampling with live-points," 2006.

[20] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "Simflex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, pp. 18–31, 2006.