Brief Announcement: Operational Analysis of Processor Speed Scaling

Kai Shen
University of Rochester
kshen@cs.rochester.edu

Alex Zhang
Hewlett-Packard Laboratories
alex.zhang@hp.com

Terence Kelly
Hewlett-Packard Laboratories
terence.p.kelly@hp.com

Christopher Stewart
University of Rochester
stewart@cs.rochester.edu

ABSTRACT

This brief announcement presents a pair of performance laws that bound the change in aggregate job queueing time that results when the processor speed changes in a parallel computing system. Our laws require only lightweight passive external observations of a black-box system and they apply to many commonly employed scheduling policies. By predicting the application-level performance impact of processing speed adjustments in parallel processors, including traditional SMPs and now increasingly ubiquitous multicore processors, our laws address problems ranging from capacity planning to dynamic resource allocation. Finally, our results show that *operational analysis*—an approach to performance analysis traditionally associated with commercial transaction processing systems—usefully complements existing parallel performance analysis techniques.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Management, performance, theory

Keywords

Multi-processor, multicore, operational analysis, capacity planning, power, P-states, ACPI, dynamic resource allocation, performance modeling, queuing, scheduling, Internet servers, datacenter-on-chip

1. INTRODUCTION

Multicore processors and other contemporary technology trends promise to revolutionize hardware and software architectures [12]. For example, they will enable consolidated "datacenter-on-chip" deployments of commercial applications that are locally distributed across clusters today [9, 17, 18]. Unfortunately, these technology trends together with the increasing complexity and opacity of modern applications threaten to obfuscate application-level performance and its relation to important system parameters in emerging parallel computing systems. Ubiquitous multicore processors make it increasingly important to understand application-level performance in parallel computing in order to make principled tradeoffs between performance and other considerations (e.g., hardware cost and power).

SPAA'08, June 14–16, 2008, Munich, Germany.
ACM 978-1-59593-973-9/08/06.

This brief announcement outlines an approach to parallel performance analysis that relates application-level performance to processor speed. Our method is based on *operational analysis* that requires only directly measurable ("operational") quantities, e.g., arrival and departure times. Operational analysis, popularized by Denning & Buzen [7], stands in contrast to stochastic queuing models [3] because the latter rely upon detailed probabilistic *assumptions*, some of which are fundamentally untestable [10]. The classical operational laws include Little's Law [15], the foundation of more sophisticated techniques including Mean Value Analysis [14]. We derive operational laws that bound the change in application-level queuing delays that would result if an observed workload were served by faster or slower processors. Such bounds can inform decisions ranging from hardware selection (e.g., capacity planning and online application migration in heterogeneous data centers) to power regulation (e.g., via processor speed scaling using the P-states of the ACPI interface [8]).

Our laws rest upon assumptions that are straightforward, easy to test, and often satisfied in real computing systems. They employ only lightweight passive external observations of black-box applications and are therefore practical in production environments where source code access, invasive instrumentation, and controlled benchmarking are not permitted. The derivations of our operational laws are nontrivial but the laws themselves are easy for nonspecialists to learn and to apply in a wide range of practical contexts.

For brevity we sketch only the theoretical aspect of our work here; we defer experiments, some technical details, and an extended discussion to the full version of this paper. A companion paper considers the complementary problem of predicting the application-level performance consequences of changing the number of processors available to an application, holding processor speed fixed [11].

2. SYSTEM MODEL

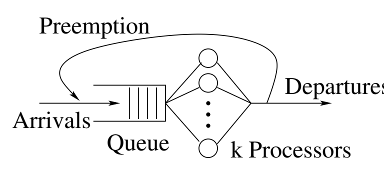

Figure 1: System model.

Consider a single-queue station with k processors as depicted in Figure 1. Jobs with heterogeneous service demands enter the system and depart after their demands are satisfied. Preemption is permitted but not required, i.e., jobs may alternate between the queue and the processors. The information available to us is severely restricted: We may observe job arrivals and departures and note the times at which they occur. Such information can be collected in practice from a network "sniffer"

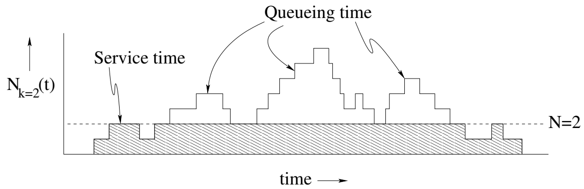

Figure 2: Service & queueing times.

monitoring an Internet server [5] or from the job dispatcher of a cluster scheduler [4, 16]. Jobs may be "anonymous" in the sense that we cannot associate specific departures with the corresponding arrivals. Furthermore the system is a black box; we have no information about the state of the queue or the processors except what we can infer from our external observations and assumptions.

ASSUMPTION 1. *Work conservation: No processor is idle unless the queue is empty.*

ASSUMPTION 2. *Serial jobs: A job occupies exactly one queue position or processor at any instant.*

Assumption 1 is satisfied by the default configuration of today's mainstream operating systems and virtual machine monitor schedulers [2, 6] and also by cluster job schedulers. Assumption 2 describes request handling in the vast majority of today's Internet servers and job processing in many compute clusters. We note that our system model is an imperfect description of computing systems in which jobs "block" (i.e., join queues other than the single queue of Figure 1). In practice, jobs may queue for mutexes and for storage and network I/O. Technology trends, however, promise to dramatically reduce blocking in emerging systems: Large main memory caches eliminate the need for blocking reads in the warm steady state and ultra-low-latency non-volatile write caches eliminate blocking for durable writes [19]. Transactional memory promises to supplant application-level mutexes [13]. Finally, whereas locally distributed application software components in a cluster make blocking remote procedure calls to one another [18], no such blocking occurs when components share a multicore processor in a consolidated "datacenter-on-chip" deployment [9]. In summary, technology trends are making our system model applicable to an increasingly wide range of practical computing systems as the multicore era unfolds.

Our observations begin with an empty system at time $t = 0$. Let $N_k(t)$ denote the number of jobs present in the system at time t, which equals the cumulative number of arrivals up to time t minus the number of departures. We call the graph of $N_k(t)$ vs. t the *occupancy curve*. The area beneath the occupancy curve equals aggregate response time across all jobs. Furthermore, as illustrated in Figure 2, the horizontal line at $N = k$ divides the area beneath the occupancy curve into aggregate service time (below $N = k$) and aggregate queueing time (above). This is because Assumptions 1 and 2 imply that the number of processors busy at any instant is the lesser of k and $N_k(t)$. The companion paper contains a detailed discussion [11]. Our goal in this paper is to bound the change in queuing time that would result if the processors were faster or slower while the workload remains unchanged. By "unchanging workload" we mean that arrival times are deterministic and do not depend on processor speed, i.e., we consider open arrivals.

3. SPEED SCALING LAWS

In practice, processing capacity can be adjusted by changing processor parameters or by migrating applications to different processors. We model such changes with a processor speedup factor: after the adjustment, the service demands of all tasks are scaled by a constant factor f. In other words, a job requiring D seconds to complete before the change will require $\frac{D}{f}$ seconds after scaling. Note that the speed scaling factor may not match the change of raw processor clock frequency since other factors like processor cache size and memory bandwidth may also affect the processing speed.

We add two assumptions about the scheduling policy and the impact of changing job arrival rate.

ASSUMPTION 3. *Deterministic scheduling: All scheduling decisions are made deterministically and they are based solely on the following information: the set of runnable jobs; arbitrary static properties of each job (e.g., priority/preemptibility); and the amount of service each runnable job has received at each scheduling decision point.*

ASSUMPTION 4. *Arrival rate monotonicity: When the arrival times of all jobs scale up by a constant factor (> 1.0), aggregate queueing time does not increase.*

Most commonly employed scheduling policies satisfy Assumption 3. In particular, it is easy to see that First-Come-First-Served, Shortest-Job-First, Shortest-Remaining-Processing-Time-First, Round Robin, and Processor Sharing are all deterministic. Assumption 4 is intuitive because the constant-factor scale-up of all jobs' arrival times means a reduction of workload intensity (less work to do) in any given time window and thus less queueing. We can show that all static priority-based preemptive schedulers (including First-Come-First-Served and Shortest-Job-First) and fine-grained Processor Sharing (or Round Robin with infinitesimal quanta) satisfy Assumption 4. For brevity we do not provide the proofs in this paper. Our first performance law provides a lower bound on queueing time reduction due to speedup.

The Processor Speedup Law. *If the k identical processors all get a speedup factor f ($f > 1.0$), then aggregate queueing up to time T will decrease by at least a factor of $\frac{f-1}{f}$. In other words, the queueing time reduction is at least $\frac{f-1}{f} \cdot A$, where A is the original aggregate queueing time:*

$$A \equiv \int_0^T \max\{N_k(t) - k,\ 0\}\,\mathrm{d}t$$

Derivation This law can be understood by two simple transformations of processing capacity and workload. In the first transformation, we scale up processor speed by a factor of f and simultaneously scale down the arrival time of all jobs by a factor of $\frac{1}{f}$. Therefore both job service time and arrival time scale down by a factor of $\frac{1}{f}$. Following Assumption 3, the same scheduling decisions will be made (albeit at different time instants) before and after the transformation. Therefore, the whole occupancy curve scales down by a factor of $\frac{1}{f}$ along the time scale (horizontal axis) after the transformation. This indicates that the aggregate job queueing time also scales down by a factor of $\frac{1}{f}$. In the second transformation, we scale back the arrival time of all jobs by a factor of f. According to Assumption 4, we know this transformation does not increase the aggregate job queueing time. Figure 3 illustrates the two transformations using a simple example of two jobs in a single-processor system ($k = 1$) running an FCFS scheduler. Each job has a service

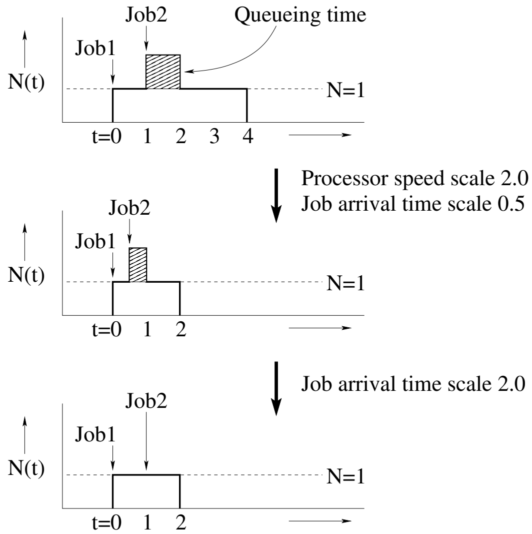

Figure 3: An illustration of the two transformations in the Processor Speedup Law derivation.

time of 2 seconds before the speedup and the processor speedup factor is $f = 2.0$. After the two transformations, we conclude that the aggregate queueing time of the jobs will scale down by at least a factor of $\frac{1}{f}$ (or decrease by at least a factor of $\frac{f-1}{f}$) due to the speedup. ■

Given the Processor Speedup Law we can easily bound the change in aggregate *response* time. The law states that aggregate queueing time will decrease by at least a factor of $\frac{f-1}{f}$ after the speedup. Since aggregate service time will decrease by exactly the same factor, aggregate *response time* (queueing time + service time) will be reduced by at least a factor of $\frac{f-1}{f}$.

Finally, we can derive a mirror result to the Processor Speedup Law for the case of processor slowdown ($f < 1.0$): Both aggregate queueing time and aggregate response time will increase by at least a factor of $\frac{1-f}{f}$. We call this **The Processor Slowdown Law**.

4. CONCLUSION

This brief announcement outlines a pair of operational laws that bound the queueing time changes in parallel processors when the the processor speed scales. Our laws only require blackbox-observable information and they apply to many commonly employed resource scheduling policies. To the best of our knowledge, operational analysis does not figure prominently in the literature on parallel computing [1]. However our results show that operational analysis can shed light on understanding the impact of speed adjustments in parallel processors, including traditional SMPs and now increasingly ubiquitous multicores.

Acknowledgments

This work was supported in part by the U.S. National Science Foundation grants CCF-0448413, CNS-0615045, and CCF-0621472. We thank Arif Merchant for detailed feedback on our work.

5. REFERENCES

[1] ACM Digital Library, January 2008. Extensive full-text keyword searches of all past SPAA proceedings for "operational," "Little," "Buzen," etc. yield only a handful of passing references to operational laws. Tracing back-pointers from classic papers such as [7] yields similar results.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, October 2003.

[3] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley & Sons, 1998.

[4] The Condor Project. `http://www.cs.wisc.edu/condor/`.

[5] Hewlett-Packard Corp. HP Real User Monitor, January 2008. Search for "Real User Monitor" at `http://www.hp.com/`.

[6] VMWare Corporat. VMWare ESX Server 3, January 2008. `http://www.vmware.com/products/vi/esx/`.

[7] Peter J. Denning and Jeffrey P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, September 1978.

[8] HP, Intel, Microsoft, Phoenix Technologies Ltd., and Toshiba. Advanced configuration and power interface specification (ACPI), October 2006. `http://www.acpi.info/spec.htm`.

[9] Ravi Iyer, Ramesh Illikkal, Li Zhao, Srihari Makineni, Don Newell, Jaideep Moses, and Padma Apparao. Datacenter-on-chip architectures: Tera-scale opportunities and challenges. *Intel Technical Journal*, 11(3):227–238, August 2007.

[10] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991.

[11] Terence Kelly, Kai Shen, Alex Zhang, and Christopher Stewart. Operational analysis of parallel servers, April 2008.

[12] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-way multithreaded SPARC processor. *IEEE Micro*, pages 21–29, March 2005.

[13] James Larus and Ravi Rajwar. *Transactional Memory*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2007.

[14] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.

[15] John D.C. Little. A Proof of the Queueing Formula: $L = \lambda W$. *Operations Research*, 9(3):383–387, May 1961.

[16] Platform Computing. LSF Scheduler. `http://www.platform.com/Products/platform-lsf-family/`.

[17] Christopher Stewart, Terence Kelly, Alex Zhang, and Kai Shen. A dollar from 15 cents: Cross-platform management for internet services. In *Proc. USENIX Annual Technical Conference*, June 2008.

[18] Christopher Stewart and Kai Shen. Performance modeling and system management for multi-component online services. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 71–84, May 2005.

[19] Texas Memory Systems. RamSan-400 Solid State Disk, January 2008. `http://www.superssd.com/products/ramsan-400/`.