

Scheduling in Highly Uncertain Environments

Rhonda Righter

IEOR Department, University of California, Berkeley, CA USA RRighter@IEOR.Berkeley.edu

We consider scheduling problems that arise on the Internet, where uncertainty and variability are high, and ask the following questions. How do we deal with extreme uncertainty? How might variability increase the range of scheduling options? When does more variability help us? When is it worth it to do extra monitoring to collect more information (and when not)? How can we compensate for a lack of information? We consider these questions for two particular Internet applications: router scheduling and computational grid scheduling.

Keywords: multi-processor scheduling, grid scheduling, stochastic scheduling

1 Introduction

In many manufacturing and service environments, there is little variability, and indeed, deterministic approximations often perform well in scheduling algorithms. In other situations randomness must be explicitly taken into account, but the variability can be captured in fairly simple models with well-behaved distributions. However, the Internet exhibits a whole new level of uncertainty in many dimensions. Internet traffic is highly variable in terms of heavy-tailed distributions, self-similar traffic, and long-range dependencies. Moreover, when many heterogeneous resources, each under the control of their own local users, are combined over the Internet into a grid to solve large-scale scientific and business problems, there are many sources of dynamic variability in terms of response times, resource availabilities, and computation errors. We consider the opportunities and challenges involved with scheduling in two Internet applications with high uncertainty. In terms of the standard scheduling problem of assigning and sequencing jobs on processors, for the first application, scheduling files on an Internet router, it is the jobs (files) that exhibit most of the variability, whereas for the second, grid scheduling, the variability and uncertainty is primarily due to the processors.

We first examine Internet router scheduling, in which there is a mixture of files of highly variable length that must be transferred, and file sizes are not known in advance. Indeed, the distribution of file sizes being sent over the Internet often has infinite variance [1]. If file sizes (job processing times) are variable enough, in a sense to be defined precisely later, then the usual nonpreemptive schedules such as first-come first-served can be the worst performing schedules. Fortunately, for Internet routers, and in contrast to most manufacturing and traditional service systems, preemption is a viable scheduling option, and can be done at little or no cost. Using preemption intelligently can greatly improve performance when processing time variability is high. In addition, for some scheduling disciplines that compensate for file size variability, performance may increase with increasing variability.

We also consider grid computing, in which a network of computers is integrated to create a very fast virtual computer. This computing paradigm is becoming ever more prevalent because of its ability to harvest a large amount of computing power at relatively low cost [3]. Grid computing creates a fast virtual computer from a network of computers by using their idle cycles. Although grid computing is a successor to distributed computing, the computing environments are fundamentally different. For distributed computing, resources are homogeneous and are reserved,

leading to guaranteed processing capacity. On the other hand, grid environments are highly unpredictable. The computers are heterogeneous, their capacities are typically unknown and changing over time, they may connect and disconnect from the grid at any time, and they may produce incorrect outputs [2].

The basic grid scheduling problem looks like a standard multi-processor scheduling problem, and if we had accurate estimates of response times for the different processors, the problem would reduce to a bin packing or load balancing problem. However, such response-time estimates are difficult to obtain and are inherently unreliable because of the dynamics of the system. On the other hand, the grid computing environment gives us a powerful new option for scheduling that does not exist (nor would it make sense) in manufacturing and standard service applications. This is the option of replicating jobs, i.e., sending the exact same work to two or more different processors that operate independently, and taking the results from whichever finishes first. When processing times are variable enough (again in a sense to be made precise later), replication makes sense in terms of minimizing mean flow times and/or makespans. In addition, it has other nice properties in terms of ease of implementation, low information requirements, robustness and adaptability, fairness, predictability, and synchronization.

Though we use the terms variability and uncertainty somewhat interchangeably to avoid repeating both words, the concepts are distinct. Roughly, variability is the range in possible values random variables can take (as measured by, e.g., range, variance, covariance), and while the random variables themselves may be highly variable, the measures of variability are generally precise. On the other hand, we may not know the parameters of the distribution (this is sometimes referred to as ambiguity), or even the possible distributional families of the random variables. Not knowing these distributions, or even the number of random variables involved, may be called uncertainty.

2 Internet Router Scheduling

Files transmitted over the Internet, or pages requested for download, exhibit extreme variability in terms of their file sizes (processing times), and often have infinite variance [1]. File types range from E-mail to video to software downloads, and file-size variability is large even within a file type. In addition, file sizes are often unknown. When file sizes are known, SRPT (shortest remaining processing time first) is the optimal scheduling policy for file transmissions for a single Internet router (a single server). Such a policy minimizes the number of jobs in the system path-wise [10], and hence minimizes mean response time. On the other hand, when file sizes are not known and are highly variable, a naive strategy such as FCFS (first come first served) will perform badly, and will produce infinite mean response time when the file size variance is infinite, even at low loads.

A reasonable model for the file-size distribution is a DFR (decreasing failure rate) distribution. For a random variable X , that we suppose continuous for simplicity, with distribution $F(x)$, tail distribution $\bar{F}(x) = 1 - F(x)$, and density $f(x) = F'(x)$, its failure (or hazard) rate at time (or age) t is $h(t) = f(t)/\bar{F}(t)$ (e.g., [6] or [11]). Hence, X is DFR if $h(t)$ is decreasing in t . Roughly, if a job has a DFR processing time, it is less and less likely to complete soon as you work on it. For example, as the time to complete a file transfer increases, it is more likely to be a long file (e.g. video rather than E-mail), so it is less likely to complete soon. DFR distributions naturally arise when mixing distributions, and imply a coefficient of variation of at least 1. Examples of DFR distributions are Pareto, hyperexponential, lognormal, and Weibull and gamma with $\alpha \leq 1$. If file sizes have a DFR distribution, then LAS (least attained service time first) stochastically maximizes the number of files completed by any time (and hence minimizes mean flow time), and any nonpreemptive discipline, such as FCFS, stochastically minimizes the number of files completed by any time [9]. (The reverse holds for IFR, i.e., increasing failure rate, distributions.) The LAS

discipline, also known as Foreground-Background, always works on the jobs with the smallest ages (the youngest jobs), in a processor sharing fashion if there are ties. So, for example, new arrivals preempt jobs currently being processed. When an interrupted job is resumed, it is resumed where it left off (so its remaining processing time has the same distribution as if it had not been preempted). For DFR processing times, LAS corresponds to SERPT (shortest expected remaining processing time first), so it is a means of sorting jobs by expected remaining size based on their age. The optimality of LAS for DFR processing time distributions holds for arbitrary arrival processes, i.e., regardless of the levels of uncertainty, variability, and long-range dependence in arrival times.

The LAS discipline can be difficult to implement, and will be impossible if there is no mechanism for keeping track of the ages of jobs. An alternative discipline that has some potential to sort jobs by size is the LCFS-pr (last come first served preempt resume). In this discipline the most recent arrival is always served, preemptively, and interrupted jobs are resumed where they left off. Under LCFS-pr a short job has the potential to finish before the next arrival. Under LCFS-pr no information on job ages is required; we need only maintain a queue. It seems reasonable to expect that LCFS-pr will perform well, and perhaps optimally in an appropriate sense, when processing times are NWU (new worse than used). A random variable X is NWU if

$$P\{X > x\} \leq P\{X - t > x | X > t\} \forall t,$$

and this is weaker than DFR (and also implies a coefficient of variation of at least 1). Note that “worse” is in a reliability sense; if a job’s processing time is NWU, then the remaining processing time (life time in reliability terms) of a job that has received no processing is stochastically shorter than that of a job that has received some processing already.

A relevant performance measure for an Internet router is the maximal queue length during a busy period, M . Since buffer sizes are typically finite, this measure is critical in helping us design a good buffer size. For this performance measure, more variability in processing times improves performance under LCFS-pr [5], regardless of whether processing times are DFR or NWU. Here we measure variability in the convex ordering sense. In particular, we say that for two random variables X and X' , $X' \geq_{cx} X$ if and only if $Ef(X') \geq Ef(X)$ for all convex functions f . Note that

$$X' \geq_{cx} X \Rightarrow EX' = EX \text{ and } Var(X') \geq Var(X).$$

Then, for two $M^X/G/1$ LCFS-pr queues (with batch Poisson arrivals) having generic service times S and S' , and corresponding busy-period maximal queue lengths M and M' ,

$$S' \geq_{cx} S \Rightarrow M' \leq_{st} M \Leftrightarrow P\{M' > b\} \leq P\{M > b\} \forall b.$$

This is somewhat surprising because generally we expect variability to degrade performance in queues, and indeed, this is the case for non-preemptive disciplines such as FCFS. For example, Miyazawa (1990) and Miyazawa and Shanthikumar (1991) show that for the finite-buffer $M^X/G/1/b$ queue under non-preemptive disciplines, the loss rate will be larger when service times are more variable in the convex sense.

3 Computational Grid Scheduling

Grid computing creates a fast virtual computer from a network of computers by using their idle cycles. Examples include networks such as the TeraGrid, a transcontinental supercomputer set up at universities and government laboratories and supported by NSF, ad hoc networks within universities or laboratories, and applications on the Internet that take advantage of the unused computing

capacity of idle desktop machines such as SETI@home, Folding@home, and Einstein@home. Grids are characterized by a high degree of uncertainty in processing times, machine availabilities, and output correctness, and these things are heterogeneous across computers and time.

The “processing time” of a job on a grid computer is the time from sending the job (or the input data) until receiving the output data. From an individual computer’s point of view, the grid job is typically a low priority job that will only be done when the computer would otherwise be idle, and only when the computer is on or connected to the network. Therefore, the processing time may include queuing times and down times, and the job could even be deleted from the computer’s queue before completion, for example if the computer is powered off or crashes. Because of firewalls and privacy concerns, the grid scheduler will typically have very little information about the status of an individual computer or the job assigned to it. Also, the computers in a grid are highly heterogeneous (and even include Sony PlayStations in the case of Folding@home). Finally, in some cases output data may be corrupted.

As a starting assumption, we suppose that the variability in the processing times comes from the computers and not the jobs. This will be the case when jobs are performing the same subroutine on different data, as in Single-Program-Multiple-Data (SPMD) parallel programs. SPMD programs are used in many applications, including computational fluid dynamics, environmental and economic modeling, image processing, simulation, and dynamic programming. Thus, the jobs are identical. We also assume that the processing time of the same job on different computers is independent, which is reasonable in a grid environment where different computers are at different locations and belong to different entities. We further assume that, given an appropriate environmental state, future processing times are independent of past processing times.

We ignore communication delays, which is reasonable when processing times are significantly longer than communication times, and is generally the case for applications in which grid computing makes sense. For example, with SPMD applications, only the input data needs to be transmitted.

With grid computing we have the option of job replication, i.e., transmitting the same input data to multiple computers, and taking the output from the first to be completed. We assume that if a job is replicated and finishes on one computer, there is no cost for deleting that job from other computers. This assumption again is reasonable in grid environments in which the reason one computer takes longer than another to process the same job is because it is either unavailable or busy with its own, higher priority, work. We only permit preemption or deletion of a job on a computer when that job completes on some other computer. When a job is deleted from a computer, a new job can be sent to that computer.

Suppose that, given the environmental state, processing times are NWU. That is, when a job is first assigned to a computer it has stochastically shorter processing (response) times than one that was assigned earlier. Again this is not an unreasonable assumption in a grid environment where, for example, a computer that hasn’t returned a result in a while may be turned off. For NWU processing times, maximal replication is optimal in terms of stochastically maximizing the departure process (the number of departures by any time t , and jointly across t) [4]. That is, we should essentially do each job sequentially, replicating each job on all available computers until a result from any computer is returned, and then replicating the next job. This will minimize mean flowtime and makespan.

Maximal replication has many nice properties that allow us to extend its optimality, given NWU processing times, to much more general models. For example, jobs may have precedences. Since without precedences sequential maximal replication in any order is optimal, with precedences, sequential replication in any order satisfying the precedences will still be optimal. Alternatively, jobs may be heterogeneous in terms of processing times or costs or deadlines or probabilities of producing correct results. All the single-machine scheduling results along with maximal replication

will hold, e.g., the optimality of SPT, weighted SPT, EDD (earliest due date first), or priority to jobs that are more likely to be done correctly, under appropriate conditions.

Maximal replication is extremely simple to implement, and is robust and adaptive; it balances the load without requiring any information about the computers. That is, it is optimal with or without that information. It easily adapts to the arrival of new computers to the grid, and it automatically fully utilizes the computers, even when the number of jobs is less than the number of computers. It is fair in the sense that no particular job will get “stuck” with a computer that is down, and it improves response-time predictability. This is important, for example, for setting QoS (quality of service) guarantees. Maximal replication also extends easily to permit error correction. For example, after replicating a single job on all computers we could wait for the first two computers to complete, and if their results agree, we would consider the job complete and replicate the next job. If they don’t agree, we could continue to wait for further results from other computers.

When using a maximal replication policy, it is easy to show that performance improves if processing times become more variable in the convex ordering sense. (More variable processing times lead to a smaller minimum processing time across the computers.)

4 Conclusion

We have considered two Internet scheduling problems, router and grid scheduling, that have a high degree of uncertainty. For each application there is a scheduling option (preemption and replication respectively) that is generally not available in standard manufacturing and non-IT service systems, and that helps us deal with the variability and compensate for a lack of information. Also, for schedules that address variability, increasing the variability can improve performance. For the grid application we have seen that for schedules that are appropriate in the context of high variability, there may be no advantage to collecting more information.

References

- [1] Crovella, M., Bestavros, A. (1996), Self-similarity in World Wide Web traffic: Evidence and possible causes, *Proceedings of the ACM Sigmetrics '96 Conference*, 160 – 169.
- [2] M. Dobber, R. van der Mei, and G. Koole (2006), Statistical properties of task running times in a global-scale grid environment, *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*, 150 – 153.
- [3] L. Foster and C. Kesselman, editors (1999), *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco.
- [4] G. Koole and R. Righter (2007), Resource Allocation in Grid Computing, *Journal of Scheduling*, to appear.
- [5] G. Koole, M. Nuyens, and R. Righter (2005), The Effect of Service Time Variability on Maximum Queue Lengths in Batch M/G/1 Queues, *Journal of Applied Probability* **42**, 883 – 891.
- [6] C.-D. Lai and M. Xie (2006), *Stochastic Ageing and Dependence for Reliability*, Springer, New York.

- [7] M. Miyazawa (1990), Complementary generating functions for $M^X/GI/1/k$ and $GI/M^Y/1/k$ queues and their applications to the comparison of loss probabilities, *Journal of Applied Probability* **27**, 684 – 692.
- [8] M. Miyazawa and J.G. Shanthikumar (1991), Monotonicity of the loss probabilities of single server finite queues with respect to convex order of arrival or service processes, *Probability in the Engineering and Informational Sciences* **5**, 43 – 52.
- [9] R. Richter and J.G. Shanthikumar (1989), Scheduling multiclass single server queueing systems to stochastically maximize the number of successful departures, *Probability in the Engineering and Informational Sciences* **3**, 323 – 333.
- [10] L.E. Schrage (1968), A proof of the optimality of the shortest remaining processing time discipline. *Operations Research* **16**, 687 – 690.
- [11] M. Shaked and J. G. Shanthikumar (2007), *Stochastic Orders*, Academic Press, New York.