Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

RANKING AND SELECTION IN A HIGH PERFORMANCE COMPUTING ENVIRONMENT

Eric C. Ni
Susan R. Hunter
Shane G. Henderson

School of Operations Research and Information Engineering
Cornell University
Ithaca, NY 14850, USA

ABSTRACT

We explore the adaptation of a ranking and selection procedure, originally designed for a sequential computer, to a high-performance (parallel) computing setting. We pay particular attention to screening and explaining why care is required in implementing screening in parallel settings. We develop an algorithm that allows screening at both the master and worker levels, and that apportions work to processors in such a way that excessive communication is avoided. In doing so we rely on a random number generator with many streams and substreams.

1. INTRODUCTION

Parallel computing environments are increasingly ubiquitous. Desktop or laptop computers with multi-core processors, local-area networks of stand-alone computers, cloud-computing environments with potentially enormous numbers of processors, and high-performance computing are becoming readily available to researchers and practitioners alike. As these environments become increasingly ubiquitous, so increases the need for stochastic simulation methodologies that can exploit these computing architectures.

There are two main research areas related to exploiting parallel computing in simulation. The first, and perhaps most well-known, is *parallel and distributed simulation*, where many processors are harnessed to simulate a *single* system. This work has a large and well-developed history of both theory and implementation (Fujimoto 2000). The second is for systems that can be simulated by one processor in a modest amount of time, and hence parallel independent replications are generated to estimate the desired performance metric. The method of parallel independent replications was most notably explored in Heidelberger (1988) and Glynn and Heidelberger (1991). In particular, these papers consider bias properties of different estimators for the expected value of a random variable where replications are obtained through parallel independent simulation replications across a set of processors.

Our interest lies in exploiting parallel architecture for simulation optimization (SO) algorithms, which seek to maximize a function that can only be observed with error through a stochastic simulation. The vast majority of SO algorithms developed to date are designed specifically for serial processing. Some commercial simulation products already exploit multicore capabilities, but to the best of our knowledge, there are no generally available guiding principles for designing SO algorithms for parallel environments. Since there are many versions of SO algorithms (depending on the properties of the search space) and many versions of parallel computing platforms (depending on the computer architecture), we investigate the complexities associated with the seemingly simplest of both: unconstrained SO algorithms on finite sets, assuming no topology, in high performance computing (HPC) environments.

Within the literature on unconstrained SO algorithms on finite sets, we focus specifically on developing efficient parallel versions of ranking and selection (R&S) algorithms for problems with up to several thousand systems where every system is simulated to some degree. Upon termination, R&S algorithms report a system as the estimated "best" among the contenders, together with a statistical guarantee that the reported system is, indeed, best. We limit ourselves to R&S algorithms, rather than to general SO algorithms (that focus on *search* and do not provide any kind of statistical guarantees on the quality of a reported solution) because the interplay of the desire for statistical guarantees along with the desire for computational efficiency creates interesting algorithmic tension and challenges.

We develop parallel R&S algorithms in the context of HPC environments, which can include thousands of cores. The HPC environment differs from other, less-reliable parallel architectures in that cores rarely fail, communication is fast and relatively straightforward, and memory is not typically shared between cores. These features mean that HPC environments are perhaps the most straightforward of parallel environments in which to explore stochastic simulation optimization. Further, while HPC environments may be less-available to the general public than cloud computing environments such as Amazon, our work is supported by the Extreme Science and Engineering Discovery Environment (XSEDE), an NSF-funded infrastructure for high-performance computing available to researchers worldwide. Thus access to an HPC environment is a plausible reality for many researchers.

The reliability of the HPC environment also enables us to develop a relatively simple structure for our parallel algorithms, in which one core is dedicated as the "master" and all other cores are "workers." The master assigns tasks to the workers, which the workers complete in parallel. The master also operates as a coordinator for the workers, so that each worker only communicates with the master and not with the other workers. To be available to process tasks from the workers, the master does not perform any simulation replications. In other less-reliable computing environments, this structure may not be desirable since cores — in particular, the master — may fail. However in the HPC environment, this simple structure is stable. We adopt a master-worker framework for the algorithms we develop.

Our study has similarities with Luo and Hong (2011), and indeed, that paper helped to motivate our work. Both studies involve the implementation of ranking and selection algorithms in parallel computing environments, and both employ sequential screening methods. An important difference is that we are working in a high-performance environment that affords the advantages mentioned above, whereas Luo and Hong (2011) employed a local-area network. Algorithmically, the two most important differences between their work and ours are that

1. we explore screening at both the master and worker levels, implementing screening at only the worker level, whereas they limited screening to the master level, and
2. our algorithm reduces the communication requirements and balances workload between the workers through an initial "Stage 0" of sampling.

Several authors have discussed parallel computing in stochastic simulation. Early work includes Heidelberger (1988), Glynn and Heidelberger (1990), Glynn and Heidelberger (1991), and those papers contain additional references. More recently, the survey paper Fu (2002) briefly mentions the potential of parallel computing for simulation optimization.

Our primary contribution is a careful exploration of the implications of random completion times for simulation replications within ranking and selection. In this setting, algorithms must be carefully designed to ensure statistical validity, and we discuss this issue in depth.

To fix ideas, we want to maximize $\mathrm{E}[f(i;\xi)]$ over $i \in \mathcal{S}$, where $\mathcal{S}$ is a finite set of "systems," indexed $i = 1, \ldots, |\mathcal{S}|$, ξ represents a random vector, and $\mathrm{E}[f(i;\xi)]$ is estimated by a sample mean. (One can think of ξ as the the set of uniform random variables used to drive a simulation replication, so its distribution need not depend on i.) Our goal is to implement an algorithm that delivers a certain probability of correct selection efficiently. By "efficiently" we mean that it achieves a large value of speedup (ratio of expected wall-clock running time on a single core to the expected wall-clock running time on multiple cores).

In our computing environment, we pay for cores in blocks of 12. For example, if we employ 13 cores in a calculation, we actually pay for 24 cores. Thus, within each block we would select the number of cores to minimize wall-clock completion time. We view this discretized cost structure as a secondary issue, and instead attempt to minimize wall-clock completion time.

2. PRELIMINARIES

As noted earlier, a wide variety of parallel computing architectures exist.

Assumption 1 We assume the following about the computing architecture:

(1) a fixed number $|\mathcal{C}|$ of cores are always available and do not "fail" or suddenly become unavailable;
(2) one core is designated the "master" and the others are "workers;"
(3) the workers are identical;
(4) the cores are capable of periodic communication;
(5) communication between cores is nearly instantaneous;
(6) messages join a queue for processing by the receiving core and are never "lost."

As is common in R&S literature, for the validity of our screening procedures, we assume that the output random variables can be produced as iid normally distributed replicates on a serial computing platform. That is, we are in the "multiple replications" setting and any dependence or non-normality in the output random variables or their estimators is *only* a function of the way the random variables are produced or aggregated in the parallel environment. We allow random simulation replication completion times — completion times may be the result of inherent properties in the systems under consideration, e.g., it takes longer to simulate a busy queuing system than a near-empty queuing system. Since the workers are identical, it is reasonable to expect that the simulation completion times are independent of the core task assignments.

More formally, we denote the output of the kth replication of simulating System i on core j as the random variable Y_{ijk}, where $1 \leq i \leq |\mathcal{S}|$, $1 \leq j \leq |\mathcal{W}|$, and $\mathcal{W}$ is the set of workers. The (random) time required to run the kth replication of System i on core j is T_{ijk}. We assume the following.

Assumption 2 If the simulation output data $\{(Y_{ijk}, T_{ijk}), j = 1, \ldots, |\mathcal{W}|, k \geq 1\}$ were produced on a single core, it would be straightforward to obtain iid replicates of (Y_i, T_i) for each $i \in \mathcal{S}$, where Y_i is marginally normally distributed with finite mean μ_i and finite variance σ_i^2. Further, $\mathrm{E}[T_{ijk}]$ is finite for all $1 \leq i \leq |\mathcal{S}|, 1 \leq j \leq |\mathcal{W}|, k \geq 1$.

3. IMPLICATIONS OF RANDOM COMPLETION TIMES

The first method one may consider when attempting to parallelize an R&S algorithm is to farm out the replications to the processors, and screen the systems as the replications finish, using naïve estimators of the means. As it turns out, there are several subtle complications with algorithms constructed in this fashion. We now discuss these complications in a simplified setting.

3.1 One System, Two Workers

For the moment, suppose that we only wish to obtain n iid replications from a single system, and to share the simulation task across two workers. Suppose the master instructs the workers to generate replications and return the results of each replication when complete. The master then uses the first n replications received as the desired sample, and constructs the sample mean across the n replications as the estimator of the expected value. This setting is discussed in detail in Heidelberger (1988), Glynn and Heidelberger (1991), and here we summarize its importance for our R&S setting.

The first n replications received by the master are more likely to have small computation times rather than large ones, so the values received are *not* the desired iid sample, even though each worker can itself generate such samples. To illustrate the bias inherent in this method, we present the following example.

Example 1 Suppose each worker $j = 1, 2$ can independently generate iid replications $Y_{j1}, Y_{j2}, \ldots$, with associated generation times $T_{j1}, T_{j2}, \ldots$. In other words, the sequence of replications completed by Worker j, $W_j = ((Y_{j1}, T_{j1}), (Y_{j2}, T_{j2}), \ldots)$, consists of iid pairs for each $j = 1, 2$, and the two sequences W_1 and W_2 are independent. (We drop the index i associated with systems, since for the moment we only have one system.) Such realizations may be obtained through the use of a random number generator with many streams and substreams, as in L'Ecuyer et al. (2002), using a different stream for each worker, and using a new substream whenever a new replication is desired.

Suppose that the first replication from Worker 1 has the same distribution as the first replication from Worker 2, as would arise if we used the same code on identical cores. Let the joint distribution of the first replication from Worker j, (Y_{j1}, T_{j1}), be such that Y_{j1} is (marginally) normal$(0,1)$, and let

$$T_{j1} = \begin{cases} 1 & \text{if } Y_{j1} < 0, \\ 2 & \text{if } Y_{j1} \geq 0, \end{cases}$$

for $j = 1,2$. Hence it takes twice as long to generate larger values as smaller values. Let T_{*1} be the time at which the master receives the first replication, or replications in the event of simultaneous arrivals. Due to the marginal normality of $Y_{j1}, j = 1,2$, we have

$$\underbrace{P(Y_{11} < 0, Y_{21} < 0)}_{T_{*1}=1} = \underbrace{P(Y_{11} < 0, Y_{21} \geq 0)}_{T_{*1}=1} = \underbrace{P(Y_{11} \geq 0, Y_{21} < 0)}_{T_{*1}=1} = \underbrace{P(Y_{11} \geq 0, Y_{21} \geq 0)}_{T_{*1}=2} = 1/4. \qquad (1)$$

Now consider the expected value of the first replication(s) received by the master. Let N^- and N^+ be random variables whose distribution is the same as $Y_{j1} \mid Y_{j1} < 0$ and $Y_{j1} \mid Y_{j1} \geq 0$, respectively, $j = 1,2$. In all cases except the last in expression (1), the first replication(s) to report will be N^- because they are computed in only one time unit. Thus, the first communication received at the master is

- two iid replications of N^- after 1 time unit with probability 1/4,
- one replication of N^- after 1 time unit with probability 1/2, or
- two iid replications of N^+ after 2 time units with probability 1/4.

The expected value of the first communication received at the master (where this value is assumed to be the average of the values of two replications if they are received simultaneously) is therefore

$$\frac{3}{4}E(N^-) + \frac{1}{4}E(N^+) = \frac{1}{2}E(N^-) < 0,$$

reflecting a negative bias, so that the first replication received is *not* distributed as Y_{11}. A similar problem arises if we average the replications that are received after any deterministic amount of time. For example, if we wait two time units and average the results received, we obtain an average of $\frac{1}{12}E(N^-) < 0$. $\square$

In the context of R&S, the screening and selection procedures we use rely on obtaining iid sequences of random variates with the correct distribution. Unfortunately, if results are returned as they complete, as in the above examples, then we do *not* obtain the correct distribution as evidenced by the presence of bias. In other words, the distribution of responses received differs from the desired iid normal sequence, despite the fact that each worker generates such a sequence.

3.2 Two Systems, Two Workers

The R&S setting we consider faces the complications discussed by Heidelberger (1988), Glynn and Heidelberger (1991) and outlined in the previous section, as well as complications resulting from simulating multiple systems on multiple cores and coordinating screening across the cores.

For example, suppose we have two systems and two workers, and we would like to select the "best" among the two systems. We can allocate the processing effort under one of the following scenarios:

(i) each worker simulates separate systems, for example, Worker 1 only simulates System 1 and Worker 2 only simulates System 2;
(ii) each worker may simulate more than one system, for example, Worker 1 simulates Systems 1 and 2, and Worker 2 also simulates Systems 1 and 2.

In Scenario (i), where each worker creates an iid stream of replicates of a separate system, one must still take care. In particular, it is not true that the set of replications that have completed by a fixed time are iid with the correct distribution. Hence, it is not clear that sample means constructed over the random number of completed replications will have the desired normal distribution upon which much of screening theory depends. Scenario (ii) is more complex in the sense that when each worker can simulate both systems, the frequency of required coordination between the workers may increase to ensure that the correct random number streams are being used and the workers are not duplicating each others' efforts. Further, care must still be taken to avoid the bias issues of Section 3.1. In general, screening in the setting of a random number of completed replications requires great care and perhaps new theory.

We can allocate screening effort in three different ways:

(a) the master performs all screening activities;
(b) the workers receive messages from the master containing information about the simulation of systems on other workers, and the workers perform all screening;
(c) the master and workers share screening activities.

If the simulation replications complete quickly, each worker communicates each replication as it finishes, and the master completes all screening activities (Scenario (a)), then the master can be overwhelmed with messages and the subsequent queuing delays. This issue can slow overall computation time, as noted by Luo and Hong (2011). Therefore we find it desirable to allow the workers to complete screening tasks in addition to the master, as in Scenarios (b) and (c). However, in doing so, ensuring that the "distributed" screenings are valid is nontrivial.

4. CONSIDERATIONS FOR VALID PROCEDURES

We now explore issues specific to a parallel computing environment that should be considered to ensure that selection procedures are valid, in the sense that they achieve the desired selection guarantee. Selection procedures typically consist of up to three stages (Kim and Nelson 2006). In Stage 1, a small initial sample of size n_0 is obtained from all systems, which provides an estimate of means and variances for each system. In Stage 2, systems are screened as the run length increases. In Stage 3, we halt screening, and choose the best of the systems that survived the screening stage by obtaining a carefully considered number of replications from each surviving system. As we have already noted, an important ingredient in all stages is an iid sequence of normally distributed variates from each system. We first explore methods for constructing such an iid sequence, and then discuss issues related to screening.

4.1 Constructing an iid Sequence

There are two broad types of estimators proposed by Heidelberger (1988): (i) estimators that produce a fixed number of replications in a random completion time, and (ii) estimators that produce a random number of replications in a fixed completion time. Heidelberger (1988), Glynn and Heidelberger (1990), Glynn and Heidelberger (1991) discuss unbiased estimators of each type. Type (ii) estimators appear to be very difficult to incorporate in our context, because even if the estimators are unbiased, which is not automatically assured, it is not clear that the replications have the desired distribution. Thus we confine our attention to Type (i) estimators. Of these, we consider exclusively those that obtain a predetermined, deterministic number of replications from each worker, thereby avoiding the issues outlined in the previous section. For example, in the one-system, two-worker example of Section 3.1, we might wait until the master receives n_j replications from Worker j, $j = 1,2$. This scheme ensures that we take the first n_j terms in the iid sequence $((Y_{j1},T_{j1}),(Y_{j2},T_{j2}),\ldots)$ for each Worker j, so there is no "re-ordering" of the simulation replications by completion time, and we therefore get the correct distribution. With this strategy, the completion time of the overall calculation is random, being a maximum over j of the sum of a deterministic number of T_{jk} random variables. Type (i) estimators fit well with Stage 1 and Stage 3 of current R&S algorithms, in which we require a predetermined number of replications from each system. With this approach the master may have to leave one worker idle for some time while the other worker completes its task. As we will see though, it is possible to design algorithms that reduce the need to idle workers, which is important in fully exploiting the capabilities of parallel computing.

The desired iid sequences of random variables can be constructed through careful use of a random number generator with many streams and substreams, as we now discuss. We separately explore the coordination of random number streams in Scenarios (i) and (ii) from Section 3.2, that is, (i) when systems are dedicated to a single worker, and (ii) when systems may be simulated on more than one worker.

Under Scenario (i) where systems are dedicated to a single worker, the assignment of random number streams is straightforward. If we wish to simulate systems independently, i.e., not use CRN, then we simply assign unique streams to systems, and each worker processes replications of the system from the assigned stream. Further, there is no need to use a separate substream for each successive replication, although doing so is useful in the extensions discussed below. If, instead, we wish to use CRN in Scenario (i), then we assign all systems the *same* stream and initial substream. To improve synchronization, we can use a new substream, common to all systems, for each successive replication, as discussed in Kelton (2006). Unbiased estimators can then be constructed from the iid output by waiting for the workers to complete a predetermined number of replications.

To assist with load balancing, we may wish to split the set of replications of a single system across multiple workers. Under Scenario (ii), where systems are not dedicated to a single worker, we must proceed carefully to avoid the issues discussed in Sections 3.1 and 3.2. As in Scenario (i), we propose that each system is assigned a fixed stream, and each replication of that system is taken from a new substream. We then assign a *set* of replications (equivalently, substreams) of a single system to each worker. For a total desired number of replications n, we ensure that the workers' sets partition the numbers $\{1,2,\ldots,n\}$. After the replications are complete, the workers communicate both the results and the indices of the substreams used to the master, which reassembles the sequence of replications in the order of the substream indices.

Example 2 Suppose that we wish to simulate eight replications of System 1, distributed over two workers. We assign stream 1 to System 1, assign substreams 1, 3, 5 and 7 to Worker 1, and substreams 2, 4, 6, and 8 to Worker 2, and identify substreams with replications. Then, irrespective of the order in which complete replications are communicated to the master, as long as the observations are ordered according to the substream indices, we recover the desired iid random variables. The key requirements are that each system uses a single stream, and that the substreams used for replications are tracked.

This stream and substream management system is reminiscent of the function of a clothing "zipper" that meshes corresponding teeth together. We refer to this management system as "zipping" replications.

Definition 1 Suppose that each system is assigned a fixed stream. An n-zip of System i consists of the replications of System i corresponding to substreams $1,2,\ldots,n$. Hence, an n-zip consists of n iid replications with the desired distribution.

To implement independent sampling, we assign a unique stream to each system. To implement CRN, we assign the same stream to all systems. In both cases we ensure the substreams are managed using zipping. When systems are dedicated to a single worker, the replications reported to the master are automatically "zipped." As before, unbiased estimators can then be constructed from the iid output by waiting for a zipped sequence of the desired length.

4.2 Screening

For now, suppose that we use a screening procedure that screens between all surviving systems at each replication count n, where we use an n-zip from each surviving system, as in, e.g., Kim and Nelson (2001). In the scenario where systems are dedicated to unique workers, the n-zip is obtained naturally.

The screening procedure can be implemented in a parallel setting as follows. Workers communicate to the master the results of all replications on all systems they are assigned. The master performs any zipping needed for each system, and screens at replication n only when n zipped replications are available from all surviving systems. The master notifies workers when systems they are simulating are screened. Workers continue generating replications from systems that they believe have not been screened, passing the results to the master as each replication finishes, until notified by the master of the identity of screened systems.

If it is required to do all pairwise comparisons of surviving systems after every replication, and each worker simulates a disjoint subset of the surviving systems, then the master must at least receive all replications from all surviving systems. This could lead to a heavy communication workload, reducing, perhaps dramatically, the benefits from a parallel implementation, as noted in Kim and Nelson (2006). Indeed this difficulty was reported in Luo and Hong (2011). The master can be overwhelmed because screening can be computationally expensive relative to the effort in simulating systems. To understand why, consider a problem with $|\mathbb{S}|$ alternatives. There are $|\mathbb{S}|(|\mathbb{S}|-1)/2$ pairs to consider in the initial screening step, and this quadratic (in $|\mathbb{S}|$) amount of work is all performed at the master. This issue remains, even if systems are not dedicated to workers, because we must first zip the system replications before screening.

Hence, we want to avoid performing *all* pairwise screenings at every step, and to distribute the screening workload among the workers.

Assumption 3 For simplicity of exposition, we assume for the remainder of the paper that we employ Scenario (i) where systems are dedicated to workers, so that zipping is automatic. The ideas below extend naturally to Scenario (ii) where systems are not dedicated to workers, provided that zipping is employed. Furthermore, we adopt Scenario (b), where workers perform all screening.

4.2.1 Avoiding All Pairwise Screenings

Suppose that each worker performs screening on the surviving systems dedicated to it. The screening is likely to be more effective if the worker can use estimated-best systems from other workers in its screening, or for less communication, the estimated-best system seen by the master. Consider the case where the estimated-best system seen by the master is communicated to all workers. If the identity of the estimated-best system changes, then the master must communicate all previous replication results of the new estimated-best system to the workers, so that they can perform all of the screening steps up to the current replication to ensure validity of the screening. (If screening is performed on a strict subsequence of replications, it may be sufficient to communicate summary statistics.) This same "catch up" screening was employed in Pichitlamken, Nelson, and Hong (2006). If all systems dedicated to a worker are screened, then the worker ceases its activity and idles. Although one could perform work balancing using zipping, recall that, for simplicity, we have assumed dedicated systems to workers. Eventually at most two workers will remain simulating, and eventually one system will screen the other, and the algorithm terminates.

Is this implementation a valid screening procedure, given that not all pairwise comparisons are performed? Many screening procedures exploit the following Bonferroni argument to establish validity. For notational ease in this section, let the number of systems $|\mathbb{S}|$ be denoted by k. The incorrect selection event ICS is the event that System k, the best system, is eliminated. In a setup in which System i and System k are simulated until one eliminates the other, let A_{ik} be the event that System i eliminates System k, assuming that System k is at least $\delta > 0$ better than System i. Then

$$P(ICS) \leq \textstyle\sum_{i=1}^{k-1} P(A_{ik}).$$

To ensure correct selection, the pairwise comparisons are chosen so as to ensure that the sum above lies below some desired threshold, i.e., that $P(A_{ik})$ is small for each $i = 1, 2, \ldots, k-1$.

If a selection procedure is based on this argument, then we argue that the above implementation is, indeed, valid, even though screening is not performed between all pairs of systems. To see why, note that an incorrect selection event arises only when System k is screened out. Since screening is performed on *all* replications between any two systems that are screened against one another, from the perspective of the two systems, the screening process is statistically identical to a sequential implementation. Therefore $P(A_{ik})$ does not change in our parallel implementation.

4.2.2 Screening on a Subsequence of Replications

Even if we do not perform all pairwise screening steps, screening can still represent a tremendous amount of computation. We may therefore prefer to screen only on a subsequence of the replications.

We do not consider screening on a *random* subsequence. Whether such a procedure would be valid is unclear, because of the random screening times. For example, suppose we compare System k (the truly best system) with some suboptimal system, say System i. If we only screen at the random times when the difference between the estimated performance of the two systems is small, then we may never screen System i out. While this particular choice of random times is artificial, it nevertheless demonstrates that care is needed to ensure the validity of screening procedures when screening occurs at random times.

Accordingly, we restrict our attention to *deterministic* subsequences. For example, we want to ensure validity when we perform screening only every 10th replication from each surviving system. One can assure validity of such an approach for many screening procedures.

For example, suppose $P(A_{ik})$ is bounded using the Paulson procedure (Paulson 1964). The Paulson procedure uses the fact that an incorrect selection arises only when a certain random walk with negative drift exceeds a fixed positive level at some point, and uses a standard result to bound the probability of this exceedance occurring in finite time. If we screen on a subsequence of the replications, then an incorrect selection arises when the random walk, observed on a subsequence of transitions, exceeds the same positive level. Exceeding the level on a subsequence implies that the level is exceeded in finite time, so the upper bound on $P(A_{ik})$ remains valid. In fact, this bound remains valid even if we screen on a *random* subsequence. In any case, subsequence screening when the procedure is based on the Bonferroni argument above together with the Paulson procedure is therefore valid.

A second important example arises when $P(A_{ik})$ is bounded using the triangular continuation region proposed by Fabian (1974). Now incorrect selection arises only when a certain random walk with negative drift exits a triangular region through the upper boundary. Fabian bounds this probability, both in discrete time, and when the random walk is embedded in a Brownian motion. Several procedures based on Fabian (1974) also exploit a result (Jennison et al. 1980, Jennison et al. 1982) that extends Fabian's results to allow the random walk to be observed only on a broader class of potentially random times. The class of random times is somewhat restrictive, e.g., stopping times are not allowed.

If the screening procedure used is of the Fabian type, then one can employ the result established in Jennison, Johnstone, and Turnbull (1980) to ensure the validity of screening on a *deterministic* subsequence of replications as desired; see, for example, Hong (2006). This same argument also applies in our setting.

5. THE ALGORITHM

We now turn from exploring design principles to a specific algorithm instance under the auspices of Scenarios (i) and (b), i.e., systems are dedicated to workers, and screening is performed only by workers. Our algorithm is a parallelized version of the "unknown-variance" sequential R&S procedure (UVP) in Section 4 of Hong (2006). See Hong (2006) for a detailed proof of statistical validity of the procedure.

The algorithm includes an initial stage, Stage 0, where we learn about the simulation completion times of each system, which are subsequently used to try to balance the workload in Stage 1 when n_0 replications are obtained from each system, and which help to determine the subsequence of replications on which we screen in Stage 2. The algorithm then screens until a single system remains, so there is no Stage 3.

Stage 0 We simulate all systems for a fixed number n_{00} of replications to obtain an estimate of simulation completion times. No other statistics from this stage are subsequently used.

Stage 1 Independently of Stage 0, systems are simulated by workers to obtain variance estimates, as in the *Initialization* phase of Hong (2006). These replications in Stage 1 are used for an initial screening, which is done at the worker level.

Stage 2 The workers simulate the remaining systems, send the results of their best system (the one with the highest sample mean) to the master, receive statistics of the best systems found on other workers, and screen. Each worker screens as in the *Screening* phase of Hong (2006) on a subsequence of replications, screening among the systems it is assigned to and the best systems from other workers, until only one system remains across all workers. To help balance the workload, in the beginning of Stages 0 and 2 we perform a random permutation of systems (in case of long runtimes for some systems that are indexed closely together), and in Stages 1 and 2 we use the simulation completion times in Stage 0 to evenly allocate systems to workers.

In each stage, we dedicate each system to a unique worker and systems are simulated independently, i.e., we do not use CRN. To save on communication effort, we do not report the result of every replication to the master, but only those replications that correspond with the subsequence on which screening is performed. This helps avoid undesirable communication congestion when the message buffers (a portion of the memory dedicated to temporarily store messages being communicated between cores) have limited size or are expensive to access.

In Figure 1 we demonstrate in greater detail how the master core allocates and distributes systems, how random number streams are created and distributed together with the assigned systems to ensure independent sampling, and how simulation results are communicated between cores. For notational simplicity, we assume that the number of systems $|\mathcal{S}|$ is divisible by the number of workers $|\mathcal{W}|$, although in practice we can freely distribute the few extra systems to workers. We use the following notation for some repeatedly-used subroutines. Send(j,n) denotes the sub-process "Send configurations for the next system to Worker j; create a new random number stream and send to Worker j; send run length n to Worker j". Receive(i,n_i) denotes the sub-process "Receive configurations for System i from the master; receive the random number stream used for System i; Receive run length for System i, n_i". Collect(*info*) denotes the sub-process "Collect *info* from all workers for all systems, in the order of worker completion". Simulate(i,n,info) denotes the sub-process "Simulate System i for n replications and record *info*".

6. NUMERICAL EXPERIMENTS

In this section, we apply the master-worker algorithm proposed in Section 5 to a stylized throughput-maximization problem for which we know the optimal solution. We briefly verify the empirical probability of correct selection (PCS) before presenting results on the empirical speed-up with different choices of communication frequency, screening method and number of cores employed.

6.1 Test Problem

We consider a 3-stage flow line problem with an infinite number of jobs in front of the first station and finite buffer space in front of stations 2 and 3. Job completion times at each station are exponentially distributed. If the buffer space in front of stations 2 or 3 become full, then a completed job in the previous station is blocked from proceeding through the line. We control the expected throughput in this problem by changing the placement of B buffers and allocating R (integer-valued) units of service rates. Then to maximize the expected throughput, we solve an integer-ordered optimization problem whose decision variables are the allocation of service rates $r \in \mathbb{Z}^3$ and the allocation of buffers $b \in \mathbb{Z}^2$. An analytical solution is found by modeling each system as a discrete-time Markov chain and computing its steady-state throughput. For $B = R = 20$, the number of systems is $\binom{19}{1} \times \binom{19}{2} = 3,249$. Two systems, namely $r = (6;7;7)$, $b = (12;8)$ and $r = (7;7;6)$, $b = (8;12)$, have the highest expected throughput of 5.776. In addition, there are four other systems with expected throughput above 5.766. If the indifference-zone parameter δ is 0.01, a correct selection should conclude with any of these six systems. A complete problem description can be found in Pichitlamken, Nelson, and Hong (2006), and sample simulation code in both Matlab and C++ language is available in Pasupathy and Henderson (2011).

```
Master Core Routine
Input: List of systems $\mathcal{S}$; Stage 0 size $n_{00}$; Approx. Stage 2
       batch size $n_1$; Parameters $\delta, \alpha, \lambda, a, n_0$ (Hong 2006).
begin Stage 0: Estimating simulation completion time
    Randomly permute systems in $\mathcal{S}$;
    for each worker $j \in \mathcal{W}$ do
        Send no. of sys. to simulate, $|\mathcal{S}|/|\mathcal{W}|$, to Worker $j$;
        for $i = 1$ to $|\mathcal{S}|/|\mathcal{W}|$ do
            Send$(j, n_{00})$;
        end
    end
    Collect(simulation time for $n_{00}$ replications);
end
begin Stage 1: Estimating sample variances
    for each worker $j \in \mathcal{W}$ do
        Determine the no. of sys. to simulate $s_j$ such that
        the sum of the completion times for the next $s_j$
        systems is approx. $T_{\text{tot}}/|\mathcal{W}|$, where $T_{\text{tot}}$ is the total
        completion time across all systems from Stage 0;
        for $i = 1$ to $s_j$ do
            Send$(j, n_0)$;
        end
    end
    Collect(sample variance $S_i^2$ for all surviving sys. $i$);
    if Screen on the master level then
        Screen among the $|\mathcal{S}|$ systems;
    end
end
begin Stage 2: Screening procedure
    $\mathcal{S} \leftarrow$ systems surviving Stage 1 screening;
    Randomly permute systems in $\mathcal{S}$;
    for each worker $j \in \mathcal{W}$ do
        Send no. of sys. to simulate, $|\mathcal{S}|/|\mathcal{W}|$, to Worker $j$;
        $T_j \leftarrow$ simulation time sum for next $|\mathcal{S}|/|\mathcal{W}|$ sys.;
        for $i = 1$ to $|\mathcal{S}|/|\mathcal{W}|$ do
            $t_i \leftarrow$ simulation time for the next System $i$;
            $n_i \leftarrow \lceil (n_1 T_j |\mathcal{W}|)/(t_i |\mathcal{S}|) \rceil$;
            Send$(j, n_i)$;
        end
        for each System $i \in \mathcal{S}$ do
            Send $S_i^2$ from Stage 1 to Worker $j$;
        end
    end
    while Number of surviving systems $> 1$ do
        Wait for next Worker $j'$ to report; Receive indexes
        of sys. eliminated on Worker $j'$; Update surviving
        sys.; Receive stats. (sample mean and sample size)
        of the best system for Worker $j'$, for each batch
        simulated up to this point; Send stats. of the best
        sys. from other workers to Worker $j'$; Instruct
        Worker $j$ to continue simulation;
    end
    Report the single surviving system as best; Send a
    terminate instruction to all workers;
end
```

```
Worker Core Routine
Input: Parameters $\delta, \alpha, \lambda, a, n_0$ (Hong 2006).

begin Stage 0: Estimating simulation completion time
    Receive the number of systems to simulate, $r$;
    for $i = 1$ to $r$ do
        Receive$(i, n_i)$;
    end
    for $i = 1$ to $r$ do
        Simulate$(i, n,$simulation time for $i)$;
    end
    Report simulation times for $r$ systems to the master;
end
begin Stage 1: Estimating sample variances
    Receive the number of systems to simulate, $r$;
    for $i = 1$ to $r$ do
        Receive$(i, n_i)$;
    end
    for $i = 1$ to $r$ do
        Simulate$(i, n_i, S_i^2)$;
    end
    if Screen on the worker level then
        Screen among the $r$ systems simulated;
    end
    Report $S_i^2$ for all surviving sys. $i$ to master;
end

begin Stage 2: Screening procedure
    Receive the number of systems to simulate, $r$;
    for $i = 1$ to $r$ do
        Receive$(i, n_i)$;
    end
    for each System $i$ surviving Stage 1 do
        Receive $S_i^2$ collected in Stage 1;
    end
    while continue = true do
        for $i = 1$ to $r$ do
            Simulate$(i, n_i, \bar{X}_i)$ for one batch;
        end
        Screen among $r$ sys. using current batch stats.;
        for each batch $k$ up to the current one do
            If batch $k$ stats. available, screen the $r$
            systems simulated, against the best systems
            from the other workers, at batch $k$;
        end
        if Master sends a terminate instruction then
            continue $\leftarrow$ false;
        else if Master is ready to communicate then
            Report indexes of eliminated sys. to master;
            Report stats. of the best system for each
            batch simulated up to this point; Receive
            stats. of the best sys. from other workers;
        end
    end
end
```

Figure 1: A master-worker algorithm for parallel ranking and selection

6.2 Programming Environment and the Use of Random Number Streams

We implement our master-worker algorithm in C++, using Message Passing Interface (MPI) for communication between cores. The program is compiled and executed on Extreme Science and Engineering Discovery Environment (XSEDE)'s Lonestar HPC cluster. According to its User Guide (Texas Advanced

Computing Center 2013), the Lonestar cluster has 1,888 nodes, each equipped with two 6-core Westmere processors and 24 GB of memory and runs Linux Centos 5.5 OS. Compiled MPI programs are submitted as batch jobs through SGE (Sun Grid Engine) 6.2 which enables users to specify the number of nodes and cores used to run the said programs.

The `RngStream` interface for C++ offered in L'Ecuyer et al. (2002) is used for random number generation. To ensure that random numbers generated on different workers for different systems and replications are independent, we use the strategy discussed in Karl et al. (2010) as follows. Each time the master assigns systems to a worker, it creates a new `RngStream` object, which is 2^{64} steps from the previous one. We then use the `GetState` method to obtain an array of six unsigned integers (the "seed") representing the state of the current stream. The seed is then sent to the worker core together with other decision variables representing the system being assigned. The worker receives the system and the seed, creates a new `RngStream` object and uses a call to the `SetSeed` method to match the state of the new stream with the master. The new stream is used by the worker to simulate this particular system exclusively, and multiple substreams can be used by calling the `ResetNextSubstream` method if the system is to be simulated on the worker for multiple replications. In this way, mutual independence between streams is maintained because all streams used are initialized and distributed by the master, and communication is minimal since only the seeds need to be sent to workers.

6.3 Numerical Results

We measure the quality of our algorithm by speedup, subject to the constraint that a certain PCS is maintained. Since our algorithm is based on Hong (2006), the PCS constraint is automatically satisfied. Indeed, using $\alpha = 0.05$, in more than 2,000 macroreplications of our algorithm the empirical PCS was 93.7%. (We followed a suggestion in Section 4.3 of Hong (2006) in choosing a parameter a of the procedure that reduces the computation but also the PCS.)

We study the speedup by changing three factors in our algorithm: the number of cores used, which is the main source of speedup; the communication frequency in Stage 2, as represented by the "batch size" n_1 (the approximate number of simulation replications per system to complete before the worker communicates summary statistics to the master; smaller batch sizes represent more frequent communication); and whether screening is centralized on the master or distributed to the workers in Stage 1. The batch size n_1 is approximate because we select different batch sizes for different systems in order to balance the workload on the workers. This is done using the results of Stage 0, so the statistical validity of our procedure is not affected. We could also try centralizing screening in Stage 2, as in Luo and Hong (2011), but preliminary tests on our platform show that it performs significantly slower (with 6 cores, our implementation of Luo and Hong (2011) requires more than 1,100 seconds) so we use distributed screening in Stage 2.

Our algorithm exhibits a number of interesting trade-offs as shown in Table 1. First and foremost, the speedup associated with the increase in the number of workers employed is most significant when the number of workers is small. Indeed, for both screening methods, increasing the number of workers from 1 to 3 or 5 introduces an almost proportional speedup. However, as a consequence of an increased level of communication, the marginal speedup from using more workers generally diminishes, as expected.

Second, distributing Stage 1 screening to the worker level has two effects in opposite directions. Stage 1 screening will most likely be faster as the pairwise comparison is parallelized, but since each worker performs only a subset of screening, and collectively across the workers we only perform a strict subset of what would have been done if screening were centralized, Stage 1 screening inevitably becomes weaker as we employ more workers. Thus more systems are left for Stage 2. (Our tests reveal that centralized screening eliminates, on average, some 3150 out of 3249 systems, whereas distributed screening with 59 workers eliminates fewer than 2000). The latter effect could explain why more cores slow down the overall execution in some cases, and why distributed Stage 1 screening performs slightly worse in almost all batch size/worker number configurations. Stage 1 screening might be more effective if, as in Stage 2 screening, we share the estimated-best solutions among workers, but our current implementation does not do so.

Table 1: Wall-clock completion times (in seconds) of the master-worker algorithm using parameters $n_{00} = 20$, $\delta = 0.01$, $\lambda = \delta/2 = 0.005$, $\alpha = 0.05$, averaged over 20 macroreplications from each configuration

Stage 1	Batch	Number of worker cores employed $\lvert\mathcal{W}\rvert$ (total cores employed is $\lvert\mathcal{W}\rvert + 1$)								
Screening	Size (n_1)	1	3	5	9	11	23	35	47	59
Master	50	100.51	35.29	22.01	13.90	11.94	13.30	9.37	10.78	16.39
	100	101.17	35.59	22.11	13.73	11.90	9.89	6.64	6.22	7.14
	200	104.33	36.61	22.80	14.20	12.06	8.68	6.09	5.48	5.42
	300	107.81	38.07	23.46	14.50	12.50	8.83	6.22	5.40	5.29
	400	110.46	39.13	24.06	14.77	12.76	10.89	6.21	5.71	5.25
	500	111.57	39.48	24.76	15.19	12.61	9.26	6.36	8.76	5.13
Workers	50	100.45	35.43	23.40	14.01	12.55	14.44	10.04	10.05	12.88
	100	101.82	36.14	25.08	14.38	13.60	22.17	7.82	6.74	7.23
	200	104.50	38.25	28.04	15.81	15.91	12.42	7.94	6.59	6.56
	300	107.14	40.22	31.58	17.13	18.45	14.94	9.26	7.74	7.16
	400	110.20	42.10	34.91	18.55	20.77	18.14	10.22	8.80	8.12
	500	111.97	44.55	38.43	20.27	23.77	17.75	11.45	9.95	8.92

Third, the effects of communication frequency in Stage 2 are again twofold. It is conceivable that smaller batch sizes (higher communication frequency) implies more screening per sample taken, and eliminates inferior systems at an earlier stage. On the other hand, the amount of communication increases as we employ more workers and a bottleneck may emerge because all communication goes through the master. As a result of both effects, fixing the worker number/screening rule combination, we typically observe that the total execution time first decreases and then increases as we increase the batch size.

7. Concluding Remarks

The parallelization of R&S algorithms designed for serial platforms is a nontrivial task. Ongoing research includes matching problem features to efficient parallel algorithm designs and creating new algorithms explicitly designed to exploit parallel platforms.

ACKNOWLEDGMENTS

This work was supported by NSF grant CMMI-1200315, and by the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF grant number OCI-1053575.

REFERENCES

Fabian, V. 1974. “Note on Anderson’s sequential procedures with triangular boundary”. *Annals of Statistics* 2 (1): 170–176.

Fu, M. C. 2002. “Optimization for simulation: theory vs. practice”. *INFORMS Journal on Computing* 14:192–215.

Fujimoto, R. M. 2000. *Parallel and Distributed Simulation Systems*. New York: Wiley.

Glynn, P. W., and P. Heidelberger. 1990. “Bias properties of budget constrained simulations”. *Operations Research* 38:801–814.

Glynn, P. W., and P. Heidelberger. 1991. “Analysis of Parallel Replicated Simulations Under a Completion Time Constraint”. *ACM Transactions on Modeling and Computer Simulation* 1 (1): 3–23.

Heidelberger, P. 1988. “Discrete event simulations and parallel processing: statistical properties”. *Siam J. Stat. Comput.* 9 (6): 1114–1132.

Hong, L. J. 2006. “Fully sequential indifference-zone selection procedures with variance-dependent sampling”. *Naval Research Logistics (NRL)* 53 (5): 464–476.

Jennison, C., I. Johnstone, and B. Turnbull. 1980. “Asymptotically optimal procedures for sequential adaptive selection of the best of several normal means”. Technical Report 463, School of Operations Research and Industrial Engineering, Cornell University, Ithaca NY.

Jennison, C., I. Johnstone, and B. Turnbull. 1982. “Asymptotically optimal procedures for sequential adaptive selection of the best of several normal means”. In *Statistical Decision Theory and Related Topics III, vol. 2.*, edited by S. Gupta and J. Berger, 55—86. New York: Academic Press, New York.

Karl, A., R. Eubank, J. Milovanovic, M. Reiser, and D. Young. 2010. “Using RngStream for Parallel Random Number Generation in C++ and R”. Unpublished Manuscript.

Kelton, W. D. 2006. “Implementing Representations of Uncertainty”. In *Simulation*, edited by S. G. Henderson and B. L. Nelson, Volume 13 of *Handbooks in Operations Research and Management Science*. Amsterdam: Elsevier.

Kim, S.-H., and B. L. Nelson. 2001. “A fully sequential procedure for indifference-zone selection in simulation”. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 11 (3): 251–273.

Kim, S.-H., and B. L. Nelson. 2006. “Selecting the best system”. In *Simulation*, edited by S. G. Henderson and B. L. Nelson, Handbooks in Operations Research and Management Science, 501–534. Amsterdam: Elsevier.

L’Ecuyer, P., R. Simard, E. J. Chen, and W. D. Kelton. 2002. “An Object-Oriented Random-Number Package with Many Long Streams and Substreams”. *Operations Research* 50 (6): 1073–1075.

Luo, J., and L. J. Hong. 2011. “Large-scale ranking and selection using cloud computing”. In *Proceedings of the Winter Simulation Conference*, WSC ’11, 4051–4061: Winter Simulation Conference.

R. Pasupathy and S. G. Henderson 2011. “SimOpt”. http://www.simopt.org.

Paulson, E. 1964. “A sequential procedure for selecting the population with the largest mean from k normal populations”. *Annals of Mathematical Statistics* 35 (1): 174–180.

Pichitlamken, J., B. L. Nelson, and L. J. Hong. 2006, 8/16. “A sequential procedure for neighborhood selection-of-the-best in optimization via simulation”. *European Journal of Operational Research* 173 (1): 283–298.

Texas Advanced Computing Center 2013. “TACC Lonestar User Guide”. Accessed May. 18, 2013. https://www.xsede.org/tacc-lonestar.

AUTHOR BIOGRAPHIES

ERIC C. NI is a Ph.D. student in the School of Operations Research and Information Engineering at Cornell University. He received a B.Eng. in Industrial and Systems Engineering and a B.Soc.Sci. in Economics from the National University of Singapore in 2010. His research interests include simulation optimization, emergency services and queuing theory. His email address is cn254@cornell.edu.

SUSAN R. HUNTER is a postdoctoral associate in the School of Operations Research and Information Engineering at Cornell University. She will join the faculty of the School of Industrial Engineering at Purdue University in the fall of 2013. Her research interests include Monte Carlo methods and simulation optimization. Her email address is hunter@cornell.edu, and her webpage is http://people.orie.cornell.edu/srh227/.

SHANE G. HENDERSON is a professor in the School of Operations Research and Information Engineering at Cornell University. He holds a B.Sc. (Hons) from the University of Auckland and an M.S. (Statistics) and Ph.D. (Operations Research) from Stanford University. His research interests include discrete-event simulation and simulation optimization, and he has worked for some time with emergency services. He is an associate editor for both *Management Science* and *Stochastic Systems*, and he co-edited the Proceedings of the 2007 Winter Simulation Conference. His web page is http://people.orie.cornell.edu/∼shane.