Parallel Independent Replicated Simulation on a Network of Workstations

Yi-Bing Lin
Bellcore
Morristown, New Jersey

Abstract

Parallel independent replicated simulation (PIRS) is an effective approach to speed up the simulation processes. In a PIRS, a single simulation run is executed by multiple computers in parallel. The statistical properties for a PIRS may be affected by the scheduling policies. For an unbiased PIRS scheduling policy, a reliable distributed computing environment is required. We consider an unbiased PIRS scheduling policy on a distributed platform such as a network of workstations. We observe that including more computing resources may degrade the performance of PIRS. Simple rules are proposed to select processors for PIRS.

1 Introduction

Discrete event simulations are often time-consuming. In many applications (such as communication network simulations), simulation processes may take several days before accurate estimates can be obtained. Recently, parallel simulation, or the execution of a single realization of a stochastic process on multiple cooperating processors, has been intensively studied to speed up the simulation process. An excellent survey of parallel simulation can be found in [2]. Bhavsar and Isaac [1], and Heidelberger [4] proposed another approach to reduce the time complexity of simulation: concurrently running multiple independent replications of the model on multiple processors and averaging the results at the end of the runs. This approach is referred to as *parallel independent replicated simulation* (PIRS). PIRS is much simpler than the parallel simulation approach, and is appropriate for stochastic simulations where the initialization bias is not severe. Properties of PIRS are studied in [3] and the references therein.

Independent replicated simulation is also a popular technique used in sequential discrete event simulation. In a long replication, the output observations are usually correlated and hence, do not satisfy the independence assumption which underlies standard statistical methods. The independent replicated simulation is considered as a simple yet effective approach to achieve independence [9]. A possible implementation

---

$i = 0$;
$\theta_\alpha = \mu = 0.0$;
while $i < 2$ **or** $\theta_\alpha > \delta\mu$ **do**
  $i = i + 1$;
  execute the ith replication, and obtain the output μ_i;
  update μ and θ_α;
end while

Figure 1: Independent Replicated Simulation.

of the independent replicated simulation is illustrated in Figure 1.

In Figure 1, at the ith iteration of the algorithm, $\mu = \frac{1}{i}\sum_{1 \le k \le i} \mu_k$, where μ_k is the output obtained in the kth replication, and θ_α is the confidence interval of α confidence level based on i samples $\mu_1, ..., \mu_i$, where $i \ge 2$. In the termination test of the while loop, both α and δ are pre-defined constants. A larger α and a smaller δ will yield a more accurate result. When the termination condition is satisfied, μ is the final output obtained. Note that termination rules other than the one illustrated in Figure 1 can be used to test termination. A very simple rule, "terminate when $i > N$" (where N is a constant), was used in [4]. The termination rule

$$\text{terminate if } \theta_\alpha \le \delta\mu \tag{1}$$

is the most popular one used in the independent replicated simulation.

In a PIRS, the while loop in Figure 1 is executed in parallel. There are several policies to schedule the processors for replication executions. The scheduling policy may significantly affect the time complexity and the statistical properties of PIRS. Different scheduling policies were studied in [1] and [4]. This paper studies the end effect of an unbiased PIRS scheduling policy on a network of workstations. Experimental study of PIRS has been conducted by Rego and Sundream [7]. They reported several important results

regarding PIRS. In their study, they assumed identical processors devoted to the PIRS applications. This paper assumes that the workstations are shared by other applications, and the workloads at workstations may change dynamically. Our study indicates that including more processors may increase the time complexity for PIRS. Several rules are proposed to select processors for PIRS.

2 Scheduling Policies

A simple scheduling policy for PIRS is the following: Suppose that $N = MP$ replications are to be executed on P processors. Every processor is assigned M replications, and one waits until all processors finish the executions. This scheduling policy is referred to as *FRP (fixed number of replications per processor)* policy. There are several problems about this policy:

- One must determine N before the replications are executed. However, it is not possible to select N which satisfies the termination condition such as (1) before we start executions.
- The processors that have finished the assigned replications are idle, and cannot be used to execute replications assigned to other processors.

A better approach is to dynamically assign replications to processors where a scheduler is required to coordinate the executions.

There are several possible scheduling policies for dynamic replication assignment. The simplest one is described as follows. The scheduler first assigns a replication to every processor. When a processor p finishes the execution, it returns the result (the output measure) to the scheduler. The scheduler then checks if the results collected so far satisfies the termination condition. If so, abort the executions of all processors and terminate. Otherwise, initiate the execution of a new replication at processor p. Assume that the termination condition is satisfied after N replication results are obtained. This scheme is referred to as the *FNC (first N replications completed)* policy because the PIRS terminates after the first N results are obtained. (Note that the first N replications completed are not necessarily the first N replications initiated.) FNC does not have the disadvantages of FRP. First, different termination rules can be used, and the termination condition is checked dynamically. Second, a processor is never idle if the termination condition is not satisfied.

FNC collects the results form the first N completed replications (if the termination condition is satisfied after the scheduler receives the N results). In other words, this scheme tends to collects the results from replications with shorter execution times. A potential problem is that for a replication, the execution time usually correlates to the output measure obtained, and collecting results from replications with short execution times may result in wrong answer. Heidelberger [4] showed that the expected value of the output measure obtained by FNC is guaranteed to converge to the wrong value unless $(N/P) \rightarrow \infty$. Consider an example where the execution time of a replication is a random variable with exponential distribution, and the correlated coefficient of the output measure and the execution time is 1. We found that for $N = 200$, the result obtained for FNC is about 40% lower than the true value when $P = 100$, and is 10% lower when $P = 20$ [6].

A scheduling policy was introduced [1, 4] to fix the bias caused by FNC. This scheme is basically the same as FNC except that every replication is assigned a number which represents the order in which it is initiated. That is, replication r_i is initiated earlier than replication r_j if and only if $i < j$, where i is the order number for r_i (note that r_i and r_j may be assigned to different processors). To compute the termination condition such as (1), the scheduler cannot use the result obtained from r_i unless it has received results from r_k for all $k < i$. This scheme is referred to as the *FNI (first N replications initiated)* policy. Suppose that this scheme collects N results, then these N results are from the first N replications initiated in the PIRS. Since the results collected are independent of the replication execution times, the final result is not biased in FNI.

In the remainder of this paper, we consider PIRS with the FNI policy.

3 Implementing PIRS on A Network of Workstations

At Bellcore, a PIRS/FNI environment was developed [6] on a network of workstations where the workstations are shared by several different applications. Two issues must be considered in such a computing platform.

Fault tolerance For PIRS/FNI, the result of replication r_i cannot be used to test the termination condition (1) if the result of replication r_k, where $k < i$, has not been received by the scheduler. If the workstation which executes r_k crashes, then the scheduler will never announce termination.

Load balancing Since the workstations are shared with other applications, the workloads on the workstations are different. A workstation with heavy workload should not be included to run replications for two reasons. First, executing a replication on a slow workstation may significantly increase the workload which affects the performance of other applications on that workstation. Second, adding an extra slow workstation may increase the time complexity of PIRS (to be studied in the next sections).

Thus, a practical implementation for PIRS/FNI has to constantly monitor all the workstations in the network and be able to dynamically migrate simulation tasks when workstation failure or slowness is detected.

The Bellcore PIRS/FNI environment was implemented based on PCI (parallel command interpreter) shell [8]. The PCI shell allows the user to program a parallel replicated simulation application in a very high level language while automatically managing the complexity caused by the dynamics of a network of workstations. PCI is able to achieve the following desired properties in PIRS/FNI:

- Load balancing: PCI automatically selects workstations with low workload for execution. During task executions, load monitoring and task migration are done automatically without user intervention.
- Fault tolerance: workstation failure is automatically detected by the PCI shell and the task can be restarted.
- Heterogeneity: The workstations selected for execution can be of different types.
- Transparency: In the PIRS environment, the user only needs to provide simulation programs, and specify termination rules. Little or no modification to an existing simulation program is required. Also, several termination rules are already built in the environment. The user only needs to select the appropriate one without any programming effort.

In the remainder of this paper, we study how load balancing and system availability affect the performance of PIRS/FNI. It is difficult to study the above issues under real workload. Instead, an artificial environment is created for analytic analysis and a Monte Simulation simulation approach similar to the one in [4] is used. In the simulation experiments, the execution time of a replication is either exponentially distributed, normally distributed, or uniformally distributed. Since the analytical analysis assumes exponential execution times, we only present the simulation results of the exponential replications for the comparison purposes (the simulation experiments with other distributions have the similar behavior as the exponential distribution). We have performed experiments using PCI. The results are briefly discussed in the Summary section.

4 The Effect of Slow Processors

Suppose that the termination condition is satisfied after N replications have been completed in PIRS/FNI with P identical processors (i.e., the processors do not execute other applications). The execution times of replications are i.i.d. exponentially distributed random variables with mean 1. Bhavsar and Isaac [1] showed that when $N >> P$, the time complexity is

$$T = \frac{N}{P} + H_P \quad \text{where} \quad H_P = \sum_{1 \le i \le P} \frac{1}{i} \qquad (2)$$

The time complexity T is affected by the *end effect* described as follows.

The End Effect. For the last P replications executed in the PIRS/FNI the PIRS has to wait until the replication with the longest execution time completes.

Note that the end effect does not occur in PIRS with the FNC policy, and the time complexity is N/P (cf., [1]). The second component H_P in (2) is caused by the end effect. Suppose that the last replication is initiated at time τ. Then the completion time of the PIRS is determined by the longest residual execution time of the P processors. Since the execution times for replications are i.i.d. random variables T_i $(1 \le i \le P)$ with an exponential distribution (with mean 1), the residual execution times after time τ also have the same exponential distribution, and the expected value for the longest residual execution time is

$$E\left[\max_{1 \le i \le P} T_i\right] = P \int_{t=0}^{\infty} te^{-t}\left(1 - e^{-t}\right)^{P-1} dt = \sum_{1 \le i \le P} \frac{1}{i} = H_P$$

Now, we extend (2) by considering the processors with different speeds. Suppose that the replication execution time T_i at processor i is exponentially distributed with mean β_i, where $\min_{1 \le i \le P} \beta_i = 1$. Then the expected elapsed time for PIRS is approximated by

$$T = \frac{N}{(1/\beta_1) + (1/\beta_2) + \cdots + (1/\beta_P)} + E\left[\max_{1 \le i \le P} T_i\right] \qquad (3)$$

The first component of (3) represents the expected execution time without the end effect, and the second component represents the extra time complexity caused by the end effect. Note that the end effect disappears as $N \to \infty$. Practically, $2 \le P \le 100$, $100 \le N \le 2000$, and we expect that the end effect has significant impact on the time complexity of a PIRS.

Suppose that a PIRS/FNI is running on a network of reliable heterogeneous workstations (i.e., the workstations never fail), and all workstations are devoted to the PIRS. In such an environment, the processor speeds may be different because the types of workstations are different. It is important to select the workstations to execute the replications. A slow workstation should not be selected because adding an extra slow workstation may increase the time complexity of PIRS. This phenomenon is caused by the end effect and is demonstrated by the following example. Consider a PIRS with P processors where

$\beta_1 = \beta_2 = \cdots = \beta_{P-1} = 1$ and $\beta_P = \beta > 1$. In other words, there are $P-1$ fast processors and one slow processor. From (3), we have

$$T = \frac{N}{P - 1 + (1/\beta)} + T_E \tag{4}$$

where

$$\begin{aligned} T_E &= E\left[\max_{1 \le i \le P} T_i\right] \\ &= \int_{t=0}^{\infty} t \frac{e^{-(t/\beta)}}{\beta} \left(1 - e^{-t}\right)^{P-1} dt \\ &+ (P-1) \int_{t=0}^{\infty} t e^{-t} \left(1 - e^{-t}\right)^{P-2} \left(1 - e^{-(t/\beta)}\right) dt \\ &= \sum_{i=0}^{P-1} \binom{P-1}{i} \frac{(-1)^i (1/\beta)}{((1/\beta) + i)^2} \\ &+ (P-1) \sum_{i=0}^{P-2} \binom{P-2}{i} (-1)^i \left[\frac{1}{(1+i)^2} - \frac{1 + (1/\beta)}{(1 + i + (1/\beta))^2}\right] \end{aligned}$$

Define the expected speedup S for a PIRS/FNI as

$$\mathrm{S} = \frac{\text{the expected time to execute } N \text{ replications at the fastest processor}}{\text{the expected elapsed time for PIRS/FNI}}$$

Then the speedup for PIRS/FNI with $P-1$ fast processors and one slow processor is

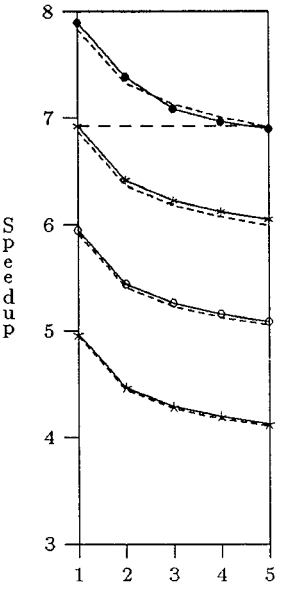
$$\mathrm{S} = \frac{N}{T} = \frac{N}{\frac{N}{P - 1 + (1/\beta)} + T_E} \tag{5}$$

Equation (5) is compared with the experimental results in Figure 2 (a) where the dashed curves represent the approximate analytical results and the solid curves represent the experimental results. The figure indicates that (5) is consistent with the experimental study. The figure also indicates that using the extra slow processor may degrade the performance. For example, when $\beta > 5$, using 7 fast processors is better than using 7 fast processors and one slow processor. Equation (5) also indicates that S increases as N increases. In other words, the end effect disappears as N increases. When $N \to \infty$, adding extra (slow) processors always improves the performance. Figure 2 (b) plots S against N (based on the experimental study).

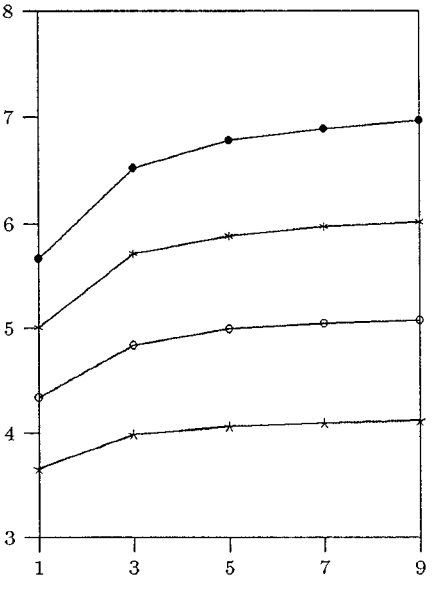
5 The Effect of Workload Changes

The previous section assumes that the speed of a workstation does not change. This section considers a network of unreliable workstations where workstations are shared by several applications. In such an environment, a workstation may not execute replications during some periods of time in two situations. In the first situation, the workstation fails and is not

Figure 2: Performance of PIRS with $P-1$ fast processors and one slow processor with workload level β ($\star : P = 5$, $\circ : P = 6$, $* : P = 7$, $\bullet : P = 8$). Solid curves: simulation. Dashed curves: analytical results.



(a) $N = 1000$



(b) $\beta = 5$

available until it is recovered. In the second situation, the workload [1] at the workstation is heavy, and executing a replication may significantly degrade the performance of other applications.

We assume that there are L workload levels for a processor. At workload level i, the mean execution time for a replication is β_i, where $\beta_1 = 1$, $\beta_i > \beta_{i-1}$ for $1 < i \leq L$, and $\beta_L = \infty$. A threshold γ is used to determine when a processor is available for executing the replications. If the workload level of a processor is higher than γ, then the processor is not available for executing the replications. If a replication is being executed when the workload changes to a level higher than γ, then the execution is interrupted, and the replication must be re-executed.

Suppose that the workload at a processor remains at a level i for a time period which is an exponentially distributed random variable with mean w (w is referred to as the *mean workload cycle time*). The workload changes from level j to level i with probability $p_{j,i}$ where $1 \leq i, j \leq L$. For simplicity, we consider the case that $p_i = p_{j,i} = p_{k,i}$ for all $j, k \neq i$. That is, the workload of a workstation changes from the current level to level i with probability p_i.

Figure 3 plots the speedup for the PIRS when $P = 57$, $L = 6$, $p_i = 1/6$ for all $i \neq j$, $\beta_1 = 1, \beta_2 = 20, \beta_3 = 30, \beta_4 = 40, \beta_5 = 50, \beta_6 = \infty$. For $\gamma = 1$, the speedup is an increasing function of w. The effect of workload change can be derived as follows. The probability that $k-1$ replications are completed before the workload changes to a level above γ (i.e., the kth replication is interrupted at the workload change) is

$$\Pr[K = k] = \int_{t=0}^{\infty} \frac{t^{k-1}}{(k-1)!} e^{-t} \frac{e^{-(\frac{t}{w})}}{w} dt = \frac{w^{k-1}}{(w+1)^k}$$

Thus, the expected number of the replications executed (including the uncompleted one) before a processor becomes unavailable is

$$\bar{K} = \sum_{1 \leq k \leq \infty} k \Pr[K = k] = 1 + w$$

Let $\mathrm{S}_{w,\gamma,N}$ be the speedup for PIRS with workload level threshold γ when the mean workload cycle time is w, and termination condition is satisfied after N replications are completed. The speedup $\mathrm{S}_{w,1,N}$ can be approximated as

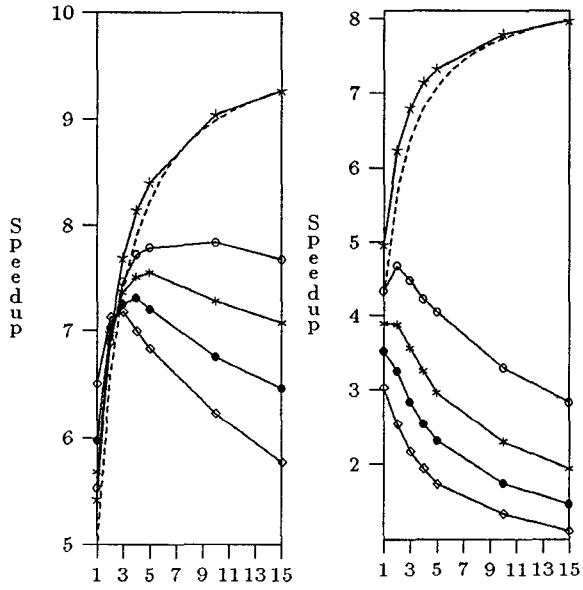
$$\mathrm{S}_{w,1,N} \simeq \frac{\bar{K} - 1}{\bar{K}} \mathrm{S}_{\infty,1,N} = \frac{1}{1 + (1/w)} \mathrm{S}_{\infty,1,N} \qquad (6)$$

The dashed curves in Figure 3 plot $\mathrm{S}_{w,1,1000}$ based on the following equation:

$$\mathrm{S}_{w,1,1000} = \frac{1 + (1/15)}{1 + (1/w)} \mathrm{S}_{15,1,1000}$$

[1] The workload does not include the load of running the simulation itself.

Figure 3: Speedups for PIRS. $P = 57, L = 6, p_i = 1/6, \beta_1 = 1, \beta_i = 10i (2 \leq i \leq 5), \beta_6 = \infty$ ($\star : \gamma = 1, \circ : \gamma = 2, * : \gamma = 3, \bullet : \gamma = 4, \diamond : \gamma = 5$). Solid curves: simulation. Dashed curves: analytical approximation.



Equation (6) indicates that for $\gamma = 1$, the speedup is an increasing function of w. It is clear that if the workload level changes very frequently, then the possibility that a replication is interrupted and re-executed increases, and the time complexity for the PIRS increases. We note that the end effect is not significant for $\gamma = 1$ when $N > 100$ (cf., Figure 4 (a)). For a large γ, the end effect is very significant for $N < 1000$, which affects the speedup curve as described below.

If γ increases, the probability that a replication is re-executed decreases. On the other hand, the end effect is more significant because a processor may execute replications at a slower rate. For a fixed γ, the probability that a replication is re-executed decreases as w increases. Equation (6) indicates that the re-execution effect is significant if w is small. (I.e., when w is less than 5, changing w may significant affect the speedup. For a large w, i.e., $w > 5$, changing w only has minor impact on speedup.) On the other hand, for $\gamma > 1$, the end effect is more significant as w increases. If the workload changes very frequently, the times to complete a replication at different processors are roughly the same, and the end effect is caused by "identical" processors. As $w \to \infty$, the end effect is determined by the slow processors with workload level γ. The previous section has already shown that

the end effect is more significant when there are slow processors. Thus, for $\gamma > 2$, when w is small, the re-execution effect is more significant than the end effect, and the speedup increases as w increases (cf., Figure 3). When w is large, the end effect is more significant, and the speedup decreases as w increases. The end effect has different impact for different γ values.

Figure 4 (a) plots speedup against N. We observe three phenomena.

- For a fixed γ, speedup is an increasing function of N. The previous section already showed that the end effect disappears as N increases.
- The speedup is less sensitive to the end effect for a small γ than a large γ. For a small γ, the speedup improvement (by increasing N from 100 to 1500) is less than the speedup improvement for a large γ. If we assume that for a fixed γ and w values the re-execution effect has the same impact on the speedup, then based on (3), we may approximate the execution time $T_{w,\gamma,N}$ for PIRS as

$$T_{w,\gamma,N} \simeq C_{w,\gamma}\left(\frac{N}{P_\gamma} + T_{E,\gamma}\right) \qquad (7)$$

$$\text{where} \quad P_\gamma = P \sum_{1 \le i \le \gamma} \frac{p_i}{\beta_i} \quad \text{and} \quad T_{E,\gamma} \simeq \beta_\gamma$$

and $C_{w,\gamma} < 1$ is the degradation factor caused by the re-execution effect. The component $T_{E,\gamma}$ is caused by the end effect (assuming that the workload level at a processor does not change when executing the last replications). Equation (7) may not give a good approximation for $T_{E,\gamma}$. However, it is clear that $T_{E,\gamma_1} < T_{E,\gamma_2}$ for $\gamma_1 < \gamma_2$ (because there are more slow processors for a large γ). The speedup improvement for threshold γ is

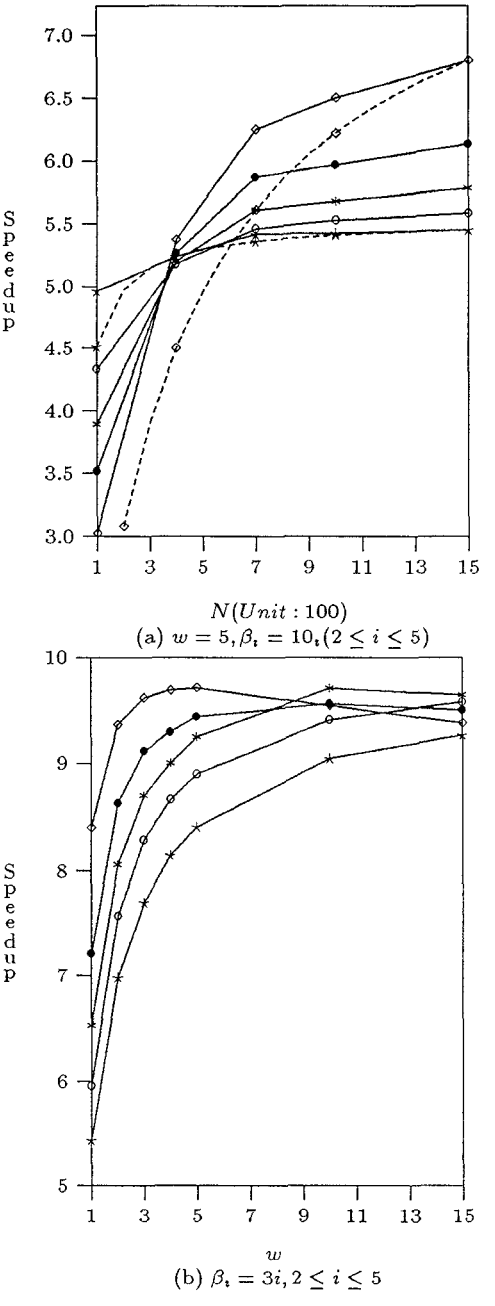
$$\frac{T_{w,\gamma,N+\Delta N} - T_{w,\gamma,N}}{T_{w,\gamma,N}} = \frac{\Delta N}{N + T_{E,\gamma} P_\gamma} \qquad (8)$$

Since $P_{\gamma_1} < P_{\gamma_2}$ and $T_{E,\gamma_1} < T_{E,\gamma_2}$ for $\gamma_1 < \gamma_2$, (8) implies that the speedup improvement for γ_1 is less than the improvement for γ_2. In Figure 4 (a), the dashed curves marked by '$\star$' (for $\gamma = 1$) and '$\diamond$' (for $\gamma = 5$) are speedup curves based on the following equation:

$$S_{5,\gamma,N} = \frac{N(1500 + P_\gamma T_{E,\gamma})}{1500(N + P_\gamma T_{E,\gamma})} S_{5,\gamma,1500}$$

Although this equation does not match the experimental results well (because the $T_{E,\gamma}$ component is not accurate), it provides a general trend of speedup changes for different γ values.

Figure 4: Speedups for PIRS. $P = 57, L = 6, p_i = 1/6, \beta_1 = 1, \beta_6 = \infty$ ($\star : \gamma = 1, \circ : \gamma = 2, * : \gamma = 3, \bullet : \gamma = 4, \diamond : \gamma = 5$). Solid curves: simulation. Dashed curves: analytical approximation.



- For a small N, the speedup for a small γ is better than the speedup for a large γ. For a large N, the result reverses. This phenomenon can also be explained by (7). For a small N, the end effect has more impact on the speedup for a large γ than a small γ. Thus, it is possible that the speedup for a small γ is larger than the speedup for a large γ. (However, if w is small, the speedup for a large γ is always better even for a small N value because the re-execution effect is very significant for a small γ). When N is large, the end effect can be ignored, and adding more processor power ($P_{\gamma_1} < P_{\gamma_2}$ for $\gamma_1 < \gamma_2$) always improves the performance.

For a fixed N, and $\gamma_1 < \gamma_2$, there may exist an w^* such that $\mathbf{S}_{w^*,\gamma_1,N} = \mathbf{S}_{w^*,\gamma_2,N}$. In Figure 3 (a), $w^* \simeq 2.5$. If w^* exists, then for $w < w^*$,

$$\mathbf{S}_{w,\gamma_1,N} < \mathbf{S}_{w,\gamma_2,N}$$

(cf., Figures 3 (a), 4 (b),and 5) This is due to the fact that for a small w, the re-execution effect has more impact on small γ, and the slow processor effect is not significant (and thus the end effect has same impact on the speedups for all γ values). For $w > w^*$,

$$\mathbf{S}_{w,\gamma_1,N} > \mathbf{S}_{w,\gamma_2,N}$$

This is due to the fact that for a large w, the re-execution effect only has minor impact for all γ values, and the slow processor effect (and thus the end effect) is more significant for a larger γ. If N is small, the impact of end effect on a large γ is so significant that the speedup for a small γ is always better than the large γ even if w is small. In such a case, w^* does not exist (cf., Figure 3 (b)).

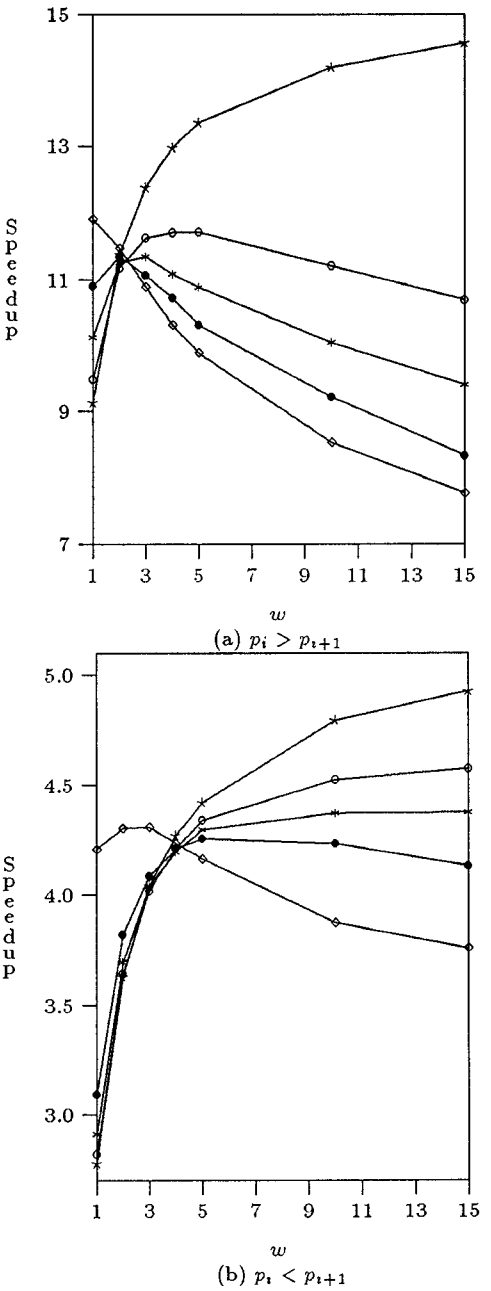
For a fixed N, if $\beta_i \simeq \beta_{i+1}$ then increasing the threshold γ from i to $i+1$ always improves the performance. In general, if (β_{i+1}/β_i) decreases, then w_i^* (i.e., the cross point w^* for the speedup curves with $\gamma = i$ and $i+1$) increases, and

$$\lim_{(\beta_{i+1}/\beta_i)\to 1} w_i^* = \infty$$

Figure 4 (b) plots the speedup curves for experiments with the same parameter setup as in Figure 3 (a), except that $\beta_2 = 6, \beta_3 = 9, \beta_4 = 12$, and $\beta_5 = 15$. The effect of (β_{i+1}/β_i) is observed by comparing Figure 3 (a) and Figure 4 (b). Since the ratio (β_{i+1}/β_i) in Figure 3 (a) is larger than the ratio in Figure 4 (b), the w^* values for the speedup curves in Figure 3 (a) are smaller than that in Figure 4 (b).

The value for w^* is also affected by p_i. In Figure 3 (a), $p_i = p_{i+1}$ and $w^* \simeq 2.5$. In Figure 5 (a), $p_i > p_{i+1}$ for $1 \le i \le 5$ and $w^* \simeq 2$. In Figure 5 (b), $p_i < p_{i+1}$ for $1 \le i \le 5$ and $w^* \simeq 3$. These figure indicate that increasing p_i for a small i decreases the w^* values.

Figure 5: Speedups for PIRS. $N = 1000, P = 57, L = 6, \beta_1 = 1, \beta_i = 10i (2 \le i \le 5), \beta_6 = \infty$. In (a), $p_1 = 0.3, p_2 = 0.225, p_3 = 0.2, p_4 = 0.125, p_5 = 0.1, p_6 = .05$. In (b), $p_1 = 0.1, p_2 = 0.125, p_3 = 0.2, p_4 = 0.225, p_5 = 0.3, p_6 = .05$ ($\star : \gamma = 1, \circ : \gamma = 2, * : \gamma = 3, \bullet : \gamma = 4, \diamond : \gamma = 5$)



If p_i is large, then a processor is likely to be in the workload level i, and it is important to set $\gamma > i$ to include enough computing resources to run the PIRS.

In summary, a small γ should be chosen if (i) N is small, (ii) w is large, (iii) (β_{i+1}/β_i) is large, or (iv) p_i is large for a small i.

6 Summary

This paper studied the performance of parallel independent replication simulation (PIRS) using a network of unreliable workstations shared by several applications. An unbiased replication scheduling policy called FNI (first N replications initiated) was considered. Our study suggests that limiting the computing resources to PIRS is necessary for two reasons: First, executing a replication on a slow workstation may significantly increase the workload which affects the performance of other applications on that workstation. Second, adding an extra slow workstation may increase the time complexity of PIRS.

We assumed that there are several workload levels for a workstation. The mean replication execution time at workload level $i+1$ is longer than the mean execution at workload level i. The workload level changes dynamically. A threshold γ of workload level is used so that PIRS does not degrade the performance of other applications too much. We found that by limiting the computing resources available to PIRS (i.e., by selecting a small threshold γ), the performance of PIRS may improve if (i) the total number of replications to be executed is small, (ii) the workload level does not change frequently, (iii) the execution times in a heavy load situation are much longer than the executions in a light load situation, and (iv) a workstation is in the light load situation more frequent than in the heavy load situation.

The above results were obtained in an artificial simulation environment (so that we can control the workload changes of workstations). We have conducted PIRS for a personal communication service network simulation [5] using PCI with 57 workstations (20 Sparc 2 workstations, 20 Sparc 1 workstations, 10 Sun 3 workstations, and 7 DEC 5000 workstations). The experiments for PIRS/FNI with workload control (a workstation is disabled if CPU utilization is higher than 70%) are compared with PIRS/FNI without workload control. One the average, the experiments with workload control are 30% faster than the experiments without workload control. We note that the workloads of workstations are different for different PIRS experiments. Although no precise conclusions can be drawn, the study does indicate the benefit of using workload control in PIRS/FNI.

References

[1] Bhavsar, V.C., and Isaac, J.R. Design and Analysis of Parallel Monte Carlo Algorithms. *SIAM Journal on Scientific and Statistical Computing*, 8:s73–s95, 1987.

[2] Fujimoto, R.M. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):31–53, October 1990.

[3] Glynn, P.W., and Heidelberger, P. Analysis of Initial Transient Deletion for Parallel Steady-State Simulation. *SIAM Journal on Scientific and Statistical Computing*, 13(4):904–922, 1992.

[4] Heidelberger, P. Discrete Event Simulations and Parallel Processing: Statistical Properties. *SIAM Journal on Scientific and Statistical Computing*, 9(6):1114–1132, November 1988.

[5] Lin, Y.-B., and Mak, V.K. Eliminating the Boundary Effect of a Large-Scale Personal Communication Service Network Simulation. *To appear in ACM Transactions on Modeling and Computer Simulation*, 4(2), 1994.

[6] Lin, Y.-B., Tseng, P., and S. Y. Hwang. Parallel Replicated Simulation and Its Extensions. *International Conference On Parallel And Distributed Systems*, pages 622–627, 1992.

[7] Rego V.J. and Sunderam V.S. Experiments in Concurrent Stochastic Simulation: The EcliPSe Paradigm. *Journal of Parallel and Distributed Computing*, 14(1):66–84, 1992.

[8] Tseng, P. Network Parallel Computing with a Command Interpreter. To appear in *International Conference on Parallel Processing*, 1992.

[9] Welch, P.D. *Computer Performance Modeling Handbook*, chapter The Statistical Analysis of Simulation Results, pages 267–329. Academic Press, 1983. Edited by S. Lavenberg.