Improving Speedup and Response Times by Replicating Parallel Programs on a SNOW

Gaurav D. Ghare   Scott T. Leutenegger
Department of Computer Science
University of Denver   Denver, CO 80208-0189
{*gghare,leut*}*@cs.du.edu*

Idle computation cycles of a shared network of workstations are increasingly being used to run batch parallel programs. For one common paradigm, the batch program task running on an idle workstation is preempted when the owner reclaims the workstation. This owner interference has a considerable impact on the execution time of a batch program, especially in the case of large parallel programs. Replication of batch program tasks has been used to reduce the impact of owner interference. We show analytically that reclamation can significantly improve parallel program speedup. Perhaps surprisingly, replication can also improve efficiency for certain workloads. We present analysis to quantify the amount of speedup and efficiency improvement. Furthermore, we provide analysis to help determine whether extra available workstations should be used for increasing job parallelism or for task replication.

I. Introduction

Networks of workstations (NOWs) have been used to run parallel programs for some time [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. The NOW may be dedicated as in the case of Beowulf [11] or shared as in the case of Condor [12], [13]. When the NOW is shared with workstation owner processes, it is referred to as a shared network of workstations (SNOW). The speedup of parallel programs running on a SNOW can be greatly affected by workstation owner interference. In this paper we consider a SNOW where parallel jobs are run in an opportunistic fashion as in Condor. In such an environment, workstation owner processes have preemptive priority over batched parallel programs. Owner process workstation reclamations stop execution of the batch job task and hence may significantly impact the parallel job response time.

One approach to prevent a workstation reclamation from impacting parallel job performance is to replicate some or all of the parallel tasks [5], [6]. When replication is used, parallel job performance is not affected by reclamations as long as one replica of each task completes without interference.

This work was supported by the NSF grant ACI-9733658.

In [5] the author compares competition protocols and migration protocols for sequential and distributed programs on varaible-speed processors. A SNOW is treated as a collection of variable-speed processors where background programs have lower priority than foreground owner programs. Competition protocols use replication to reduce the impact of owner interference. Simulation results of the performance of competition for distributed programs are presented. However there is no mathematical analysis of the performance of competition protocols for distributed programs. Competition policy issues such as, how to allocate workstations for replicating tasks of a program, are not studied.

The authors in [6] demonstrate that owner interference considerably impacts the performance of a parallel program running in a SNOW environment. Futhermore, they demonstrate that using task replication can significantly improve job response time. The study only considers the loss of up to one workstation in a batch of tasks. Tasks are assumed to be of different sizes allowing for a mix of short and long service demand tasks. Performance of replication for programs with tight coscheduling requirements is not studied. They consider one no-replication and two replication strategies to alleviate the negative impact of owner interference: no-replication (NR) adds the extra workstations to the general pool, single replication (SR) replicates the largest task in a batch using one extra workstation and uses other extra workstations in the general pool, and complete replication (CR) replicates the K largest tasks using K extra workstations. They show with experimental workloads that CR performs better than SR. Our work differs in that we provide analysis and proofs instead of simulation, we consider synchronized workloads, we study various tradeoffs in how best to allocate K extra workstations, and we consider the tradeoff between using extra processors for parallelism versus replication.

Rosenberg [10] develops a model for devising a schedule that maximizes the amount of work accomplished from an embarassingly parallel workload. Owner interference is considered as an adversial process between the owner and the user running background programs. The instantaneous probability of workstation reclamation is assumed to be known. In [2] the authors consider sharing a bag of identically complex tasks in a heterogeneous network of workstations (HNOW). The problem considered is accomplishing as much work as possible on the HNOW, during a prespecified fixed period of time. Neither of [2], [10] consider the effect of replication for reducing the impact of owner interference on parallel program performance.

In this paper we show that replication can result in significant speedup improvements and we analytically quantify parallel task replication benefits for two workload models: tightly-coupled barrier synchronized and loosely-coupled barrier synchronized. We allow for multiple workstation reclamations during the execution of a batch of tasks. We assume knowledge of the probability that a workstation may be reclaimed before a task completes, but do not assume knowledge of the instantaneous probability of workstation reclamation as in [2], [10]. We analytically study how to distribute extra workstations not only among tasks of a program but also between two programs.

In a dedicated parallel processing machine, increasing a parallel program's workstation allocation beyond the program's maximum parallelism will reduce workstation efficiency without any speedup improvement [14]. We show that this assumption does not hold in the case of a SNOW. On the contrary, and somewhat surprisingly, additional workstation allocation in the form of task replication can result in improvements in efficiency as well as speedup.

The rest of the paper is organized as follows. In Section II we describe our machine and workload models. In Section III we present replication analysis and results for both workload models considered. In Section IV we present analysis and explore the trade-off between using additional workstations for replication versus for increasing job parallelism. In Section V we prove lemmas that are used in theorems. We state our conclusions in Section VI.

II. Machine and Workload Model

In this section we explain our overall system model, specific parallel program models, and performance metrics.

A. System and Workstation Model

We assume a SNOW system of N homogeneous workstations. As in the Condor system [12], [13], we assume workstations execute both owner jobs and batched jobs. We assume batch *jobs* are parallel programs, that are decomposed into *tasks*, and then the tasks are run in parallel on idle workstations. For simplicity we assume that all the tasks are of an equal length, that owner jobs are sequential processes local to the workstation, and migration and checkpointing overheads are absorbed in the workload demand. Similarly, homogeneity of workstations is assumed for simplicity.

We assume workstation owner processes have preemptive priority over batch tasks. As soon as an owner job begins execution after an idle period, the workstation is reclaimed, and the running batch task, if any, is preempted immediately. All task work completed since the last checkpoint is lost.

We assume workstation reclamations are independently and identically distributed, and that the probability a workstation is reclaimed during the length of a task unit is p_r, $0 < p_r < 1$. When a task is preempted the task is allocated to another idle workstation and is restarted on that workstation. The partial work completed by the task is lost and the task is restarted at the beginning on the newly allocated workstation. The task only restarts after the current length of a task unit is finished. The time when one set of job tasks is deemed to be completed and the next set of tasks started may represent the point in the program where barrier synchronization takes place or when a job checkpoints.

We assume we have enough workstations at hand so when one (or more) of the workstations is reclaimed we have another one (or more) upon which to run the task(s). Thus a job always has a fixed number of workstations allocated to it. We make this assumption to simplify analysis, but it should have no effect on the qualitative results gleaned from this study.

When we say a task is replicated d times, we mean there are a total of $d > 1$ identical tasks scheduled, and the task completes when one (or more) of them finishes. However if $d = 1$, we say the task has not been replicated.

B. Parallel Job Models

The batch workload consists of parallel programs, or jobs, each with N tasks, where $N \geq 1$. A job completes when all of the N tasks have completed. For simplicity, we assume all tasks to be of unit length. Different inter-task synchronization constraints impact the job performance. We have recently begun exploration of the effectiveness of replication for a Master-Worker workload [15], but for now consider the following workload models:

- **Tightly-Coupled Barrier:**

We assume a job is composed of N equal sized tasks. A series of B barriers must be completed. A barrier is reached when each of the N tasks completes. All tasks must be simultaneously scheduled for forward progress. If any of the N tasks is preempted, all N tasks must be started over. Thus, a barrier is only achieved after a task-demand sized time interval where none of the N workstations is reclaimed.
We assume that tasks always start at fixed intervals of time. When a workstation is reclaimed, the task must be moved and hence the completed work of *all* job tasks (of the current barrier) is lost and all these tasks must be re-executed from the beginning. We also assume, for simplicity and to model task migration time, that the tasks can be restarted only at the beginning of the next interval.
We only explicitly model one barrier, since the analysis is the same for B barriers, with the response time being B times that in the case of one barrier.

- **Loosely-Coupled Barrier:**
  The model for loosely-coupled is the same as tightly-coupled with one difference: the only synchronization constraint during the time between barriers is waiting for each of the other $N-1$ tasks to reach the barrier. Thus preemption of one task does not affect the completion of the other $N-1$ tasks.
  It is necessary for all the N workstations to have finished their current task before the job (any of its tasks) can proceed to the next barrier. Again, we do not explicitly model more than one barrier, since the analysis is the same for B barriers, with the response time being B times that of one barrier.

C. Performance Metrics and Notation

Our primary metrics of interest are batch job speedup and efficiency [14]. Since we want to study the impact of replication, we define S_d to be the speedup of a job when each of its tasks is replicated d times. Likewise, R_d and E_d are the response time and efficiency assuming each task is replicated d times. In the case when the tasks of a program have different degrees of replication, d represents the relevant degree of replication being discussed. We summarize our notation in Table I.

III. Analysis and Results

For the sake of readability we state the relevant Lemmas before the Theorem statements. The proofs of the Lemmas are in section V

TABLE I: Notation

N	Parallelism of a program
d	Degree of replication
p_r	Probability of workstation reclamation
S_d	Speedup, where d is the degree of replication
R_d	Response Time, where d is the degree of replication
E_d	Efficiency, where d is the degree of replication
$F(t)$	CDF of the probability distribution
$\mu_{N:N}$	Mean of a maximum of N geometric variables
B	Number of barriers in a program, each having N tasks
f	Sequential fraction of a parallel program
T_i	Task i of a program

A. Tightly Coupled Barrier

In this section we present our analysis and results for the tighly-coupled barrier synchronization model. Assume a job has parallelism N and that the probability of a workstation reclamation during task execution is p_r. The probability that a task completes is $(1-p_r)$. The program completes a set of N tasks, if all the workstations complete the task allocated to them. Thus the probability that the job completes a set of N tasks is $(1-p_r)^N$. Assuming the program has a linear speedup on a dedicated set of workstations, the expected speedup of the program on a non-dedicated set of workstations is $S_1 = N(1-p_r)^N$.

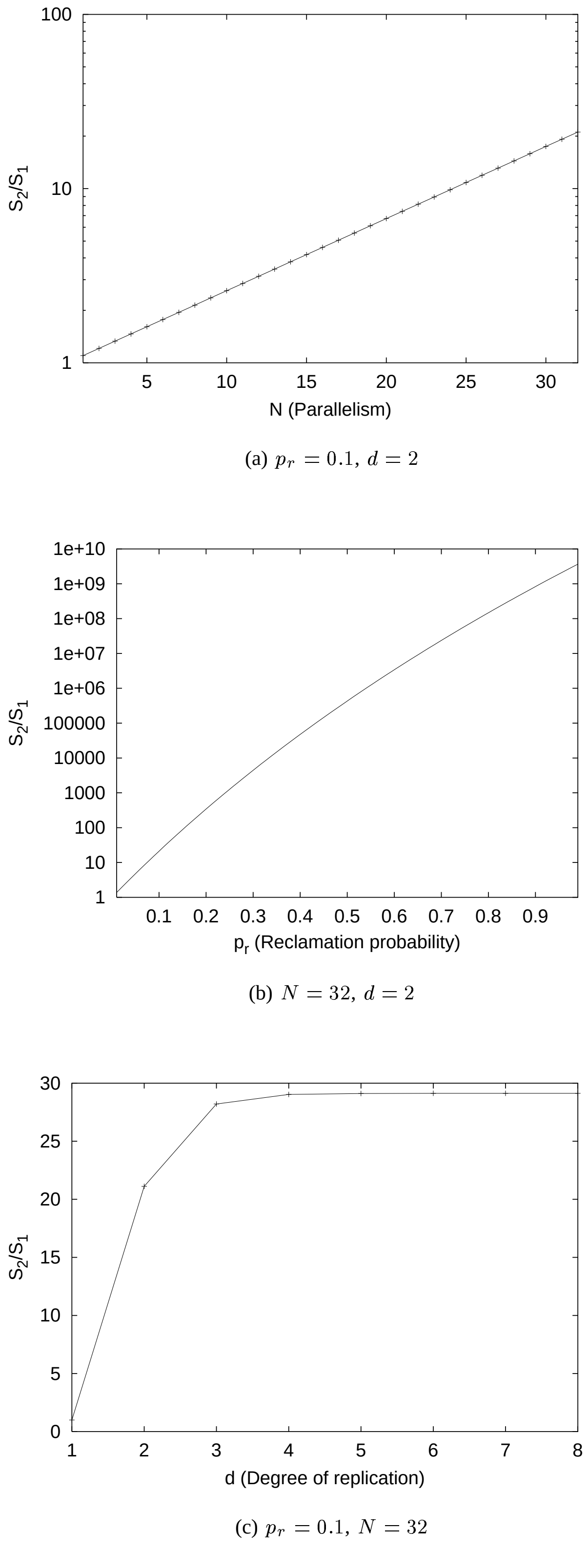
When a task is replicated on two workstations, the task gets completed when either one of them completes. The probability that the task is completed during the first allocation is $(1-p_r^2)$. If all the tasks in a program are replicated then the probability that a program completes a set of tasks is $(1-p_r^2)^N$. The expected speedup of the replicated program (assuming linear speedup) is $S_2 = N(1-p_r^2)^N$.

The ratio of speedup with replication to the speedup without replication is

$$\frac{S_2}{S_1} = \frac{N(1-p_r^2)^N}{N(1-p_r)^N} = (\frac{1-p_r^2}{1-p_r})^N$$

$0 \leq p_r \leq 1$ implies $(1-p_r^2) \geq (1-p_r)$ and the speedup ratio grows exponentially as the parallelism N is increased. The speedup with replication is guaranteed to be at least as much as the speedup without replication, if not better.

In Fig. 1 we present plots of the ratio of speedup with replication over the speedup with no replication. Unless varied, we assume p_r = 0.1, N = 32, and d (replication degree) = 2. We chose N to be 32 as a typical sized parallel program. Even though replication



(a) $p_r = 0.1,\ d = 2$

(b) $N = 32,\ d = 2$

(c) $p_r = 0.1,\ N = 32$

Fig. 1: Speedup Ratio (Replication over No-Replication)

provides a much better performance improvement for higher values of p_r, we select $p_r = 0.1$ because running tightly coupled parallel jobs on a SNOW with high owner interference may not be practical.

In all three graphs, on the y-axis we plot the ratio of speedup with replication over the speedup with no replication. In Fig. 1(a) we vary job parallelism. We see that the ratio increases with the job parallelism. This is because as N increases, the probability of a task being preempted, and all of the tasks requiring a restart, also increases. With replication, *both* a task and its replica must be preempted before a barrier needs to be re-executed. Hence, the speedup ratio increases significantly.

In Fig. 1(b) we vary p_r. We see that as p_r increases, the utility of replicating tasks increases.

In Fig. 1(c) we vary the degree of replication, d. We see that for the parameters chosen, increasing the degree of replication up to 3 significantly improves job speedup.

Replication can also increase the efficiency [14] defined as the ratio of speedup to the number of workstations, $E(n) = S(n)/n$. Assuming all the tasks of a program are replicated once, the number of additional workstations allocated to the parallel program is equal to N, where N is the parallelism of the program and is equal to the number of workstations allocated to the program without replication. The efficiency is improved if $E_2/E_1 \geq 1$ where E_2 , E_1 represent the efficiency of the program with and without replication respectively. $E_2/E_1 = S_2/(2S_1)$, so the efficiency is improved if $S_2/S_1 \geq 2$. This happens when :

$$\frac{(1-p_r^2)^N}{(1-p_r)^N} \geq 2$$

$$\frac{1-p_r^2}{1-p_r} \geq 2^{1/N}$$

$$\log_2\left(\frac{1-p_r^2}{1-p_r}\right) \geq \frac{1}{N}$$

$$N \geq \frac{1}{\log_2\left(\frac{1-p_r^2}{1-p_r}\right)}$$

Given p_r, we can calculate the parallelism for which the efficiency of the program will be improved by replicating all its tasks. We can use the knowledge of N and p_r to determine the degree of improvement (or degradation) replication causes in the efficiency.

In the general case when each task i of the program of parallelism N is replicated d_i times, where $d_i \geq 1$, the speedup of the replicated program is $N\prod_{i=1}^{N}(1 - p_r^{d_i})$. Replication causes improvement in efficiency if $N\prod_{i=1}^{N}(1-p_r^{d_i}) \geq (1-p_r)^N \sum_{i=1}^{N} d_i$.

Fig. 2 shows the minimum parallelism a program must have, to achieve improved efficiency by replication. Programs that are larger (have higher parallelism) than the minimum parallelism will have better efficiency with replication than without replication. All the tasks of a program are replicated to an equal degree.

In Fig. 2(a) we vary p_r. We plot the minimum parallelism needed when the degree of replication is 2, 3 and 5. For a low value of p_r (low owner interference), replication improves efficiency only for large programs. When the owner interference is high, replication improves performance significantly and is thus able to improve the efficiency of both large and small programs.

In Fig. 2(b) we vary the degree of replication d on the x-axis and view its effect on the minimum parallelism which is plotted on the y-axis. A lower degree of replication improves the efficiency for programs with smaller parallelism than does a higher degree of replication. Note, for higher degrees of replication where the efficiency ratio is less than 1, it is the case that higher speedup is achieved, but at the cost of a lower efficiency.

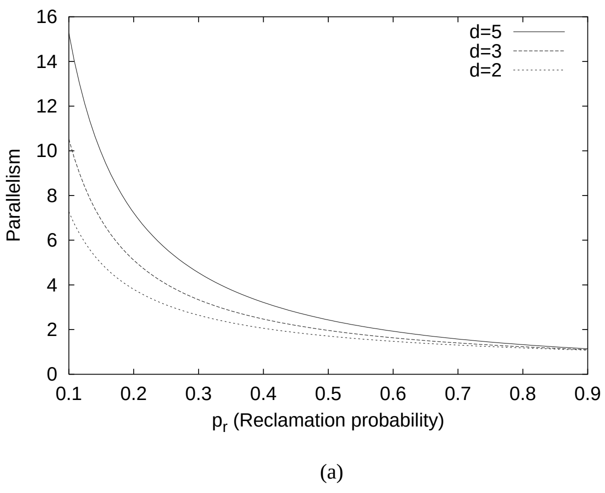
1) Allocating extra workstations to the same program with equal replication: In the previous sections we considered the benefit of replicating all of the tasks relative to no replication. In this section we investigate the best way to allocate additional replicas when we do not have enough workstations available to replicate all the tasks an additional time. Suppose we have k workstations ($1 \leq k \leq N$) to allocate to a program of parallelism N, whose tasks are all replicated d times. The question we have is : Is it better to use all these k workstations to replicate just one task of the program k additional times, or to use them to replicate k tasks of the program 1 additional time?

Lemma 5.1: If $0 < p < 1$, $n > 0$ and $k > 0$, then $\forall m > n$, we have

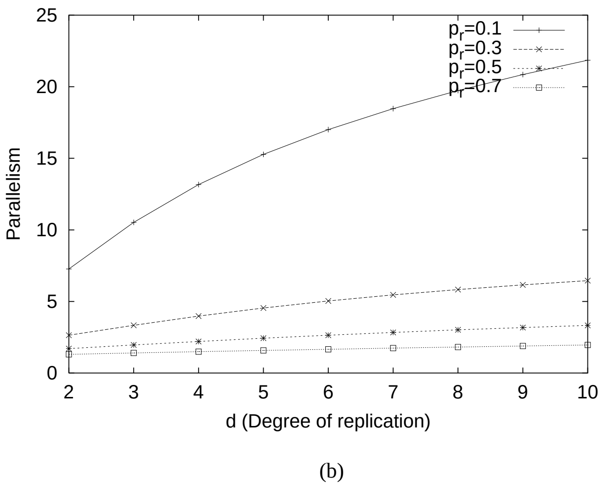
$$\frac{1-p^{n+k}}{1-p^n} > \frac{1-p^{m+k}}{1-p^m}$$

Theorem 3.1: Replicating k tasks one additional time gives better speedup (and response time) than replicating one of the tasks k additional times.

Proof: Let S_{d+k}, S_{d+1} denote the estimated speedups when one task is replicated k additional times and when k tasks are replicated one additional time respectively. We have



(a)



(b)

Fig. 2: Minimum parallelism for which replication improves efficiency

$$S_{d+k} = N(1-p_r^d)^{N-1}(1-p_r^{d+k})$$

$$S_{d+1} = N(1-p_r^d)^{N-k}(1-p_r^{d+1})^k$$

For $k = 1$, we have $S_{d+1} = S_{d+k} = N(1 - p_r^d)^{N-1}(1-p_r^{d+1})$. For $k = 2$, we have

$$S_{d+k} = N(1-p_r^d)^{N-1}(1-p_r^{d+2})$$

$$S_{d+k} = N(1-p_r^d)^{N-1}(1-p_r^{d+1})\frac{(1-p_r^{d+2})}{(1-p_r^{d+1})}$$

$$S_{d+1} = N(1-p_r^d)^{N-1}(1-p_r^{d+1})\frac{(1-p_r^{d+1})}{(1-p_r^{d})}$$

By Lemma 5.1 (Section V) we know $S_{d+1} > S_{d+k}$. Similarly, for $k > 2$, we can reapply Lemma 5.1 $k-1$ times to show that $S_{d+1} > S_{d+k}$.

Thus it is better to replicate k tasks of the program one additional time, than to replicate one task k times. ■

In Fig. 3 we present plots of the ratio of speedup (S_{d+1}) when one workstation is allocated to each of k tasks over the speedup (S_{d+k}) when all k workstations are allocated to one of the tasks. In all four graphs, we plot the speedup ratio S_{d+1}/S_{d+k}, as the y-axis. Unless varied, we assume $p_r = 0.1$, $d = 2$, $N = 32$, and the number of extra workstations to be allocated $k = 4$. Before allocation of the extra workstations, each of the tasks has an equal degree of replication.

In Fig. 3(a) we vary the parallelism N. We plot N only for values where $N \geq k$. Assume, without loss of generality, that the allocation of the k workstations is done among the first k tasks. For every value of N allocating the additional workstation evenly results in a speedup ratio of 1.03 for d = 2 and 1.32 for d = 1. From the expressions for S_{d+1} and S_{d+k}, we can see that the tasks $k+1$ through N contribute equally to the speedup in both cases. Consequently, they do not affect the ratio S_{d+1}/S_{d+k}. For $N \geq k$, the ratio of the speedups is independent of N.

In Fig. 3(b) we vary the initial degree of replication d (before allocating the k extra workstations) and study its effect on the ratio S_{d+1}/S_{d+k}. For a lower initial degree of replication, the allocation of k workstations to the program has a greater effect. As d increases, the k workstations have a smaller effect on the overall speedup. Thus the ratio of speedups approaches 1 for higher values of d.

In Fig. 3(c) we vary p_r. The benefit of allocating the k workstations is greater for higher amounts of owner interference. Thus the benefit of distributing the extra workstations among the k tasks is more pronounced for higher values of p_r.

In Fig. 3(d) we vary the number of extra workstations k. When $k = 1$ the ratio of speedups equals 1. As the number of extra workstations is increased, the difference in the performance of the two allocation policies increases.

2) Allocating an extra workstation to the same program with non-identical replication: Let us assume we have a program J with N tasks (T_1, T_2, ... T_N) running in parallel. Let us assume each task T_i has d_i replicas. If we have an extra workstation to allocate to one of these tasks, we wish to determine the allocation that will maximize speedup.

Lemma 5.1: If $0 < p < 1$, $n > 0$ and $k > 0$, then $\forall m > n$, we have

$$\frac{1-p^{n+k}}{1-p^n} > \frac{1-p^{m+k}}{1-p^m}$$

Theorem 3.2: Speedup is maximized when the extra workstation is allocated to the task that has the least replicas.

Proof: The speedup of the program before allocating the extra workstation is given by $S = N(1-p_r^{d_1})(1-p_r^{d_2})...(1-p_r^{d_N}) = N\prod_{i=1}^{N}(1-p_r^{d_i})$. Let S_j denote the speedup of the program when the extra workstation is allocated to task T_j (i.e. task T_j is replicated one additional time). $S_j = N(1-p_r^{d_j+1})\prod_{i\neq j,i=1}^{N}(1-p_r^{d_i})$. Also,

$$\frac{S_j}{S} = \frac{N(1-p_r^{d_j+1})\prod_{i\neq j,i=1}^{N}(1-p_r^{d_i})}{N\prod_{i=1}^{N}(1-p_r^{d_i})}$$

$$\frac{S_j}{S} = \frac{(1-p_r^{d_j+1})}{(1-p_r^{d_j})}$$

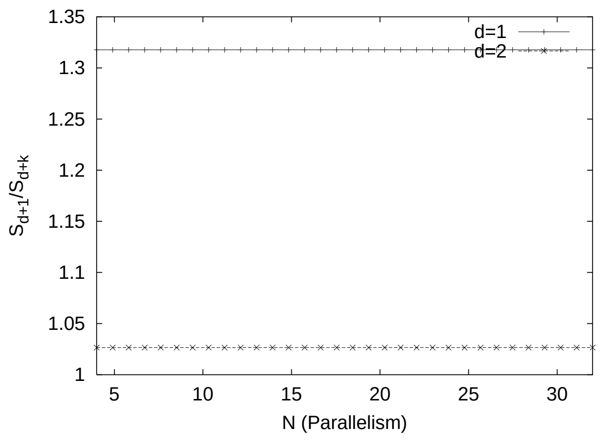
We can assume, without loss of generality, that T_1 has the least replicas, i.e. $d_1 \leq d_j$, $\forall j \neq 1$. By Lemma 5.1 we know that $S_1/S \geq S_j/S$, $\forall j$, $1 < j \leq N$. Thus $S_1 \geq S_j$. Therefore we maximize speedup by allocating the extra workstation to the least replicated task. ■

In Fig. 4 we plot the ratio of the speedup when we allocate the extra workstation to the task with degree of replication d_1 over the speedup when we allocate the extra workstation to the task with degree of replication d_j, where $d_1 \leq d_j$.

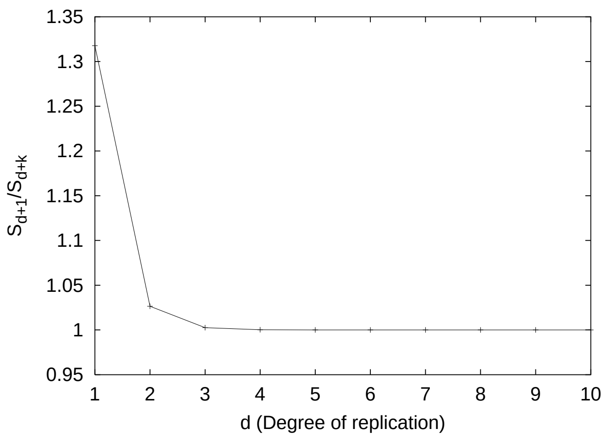
In Fig. 4(a), we vary d_1 with a fixed value of $d_j = 10$. As the difference in the initial degrees of replication between the two tasks (before the extra workstation is allocated) is low, the ratio of the speedups approaches 1. It is more important to allocate the extra workstation to the least replicated task, when the difference in their degrees of replication is high.

In Fig. 4(b), we vary p_r. At higher values of owner interference using the extra workstation to replicate the least replicated task becomes more important.

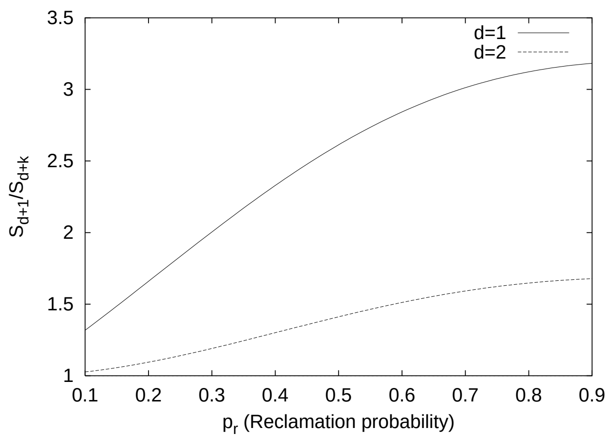
3) Allocating extra workstations to identical programs: We now consider the case where we have TWO parallel jobs, each with equal parallelism, and want to allocate additional replicas in the best way possible. Consider the problem of allocating k extra workstations



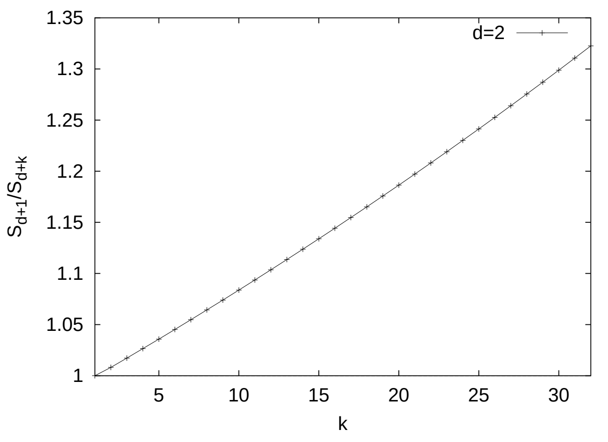
(a) $p_r = 0.1$, $d = 2$, $k = 4$



(b) $p_r = 0.1$, $N = 32$, $k = 4$



(c) $d = 2$, $N = 32$, $k = 4$



(d) $p_r = 0.1$, $d = 2$, $N = 32$

Fig. 3: Speedup ratio (One-to-each over All-to-one)

to jobs J_1 and J_2 which each have $N \geq k$ tasks. Each task of both the programs has d replicas. In other words, the parallelism and replication of each job is equal. Our objective is to allocate the additional k workstations so as to minimize mean response time.

Lemma 5.3: If $x \geq 1$, $m \geq 0$, $n \geq 0$ and wlog $m \geq n$, then $1 + x^{m+n} \geq x^m + x^n$.

Theorem 3.3: Giving some extra workstations to each program results in a better mean response time than giving all of them to one of the programs.

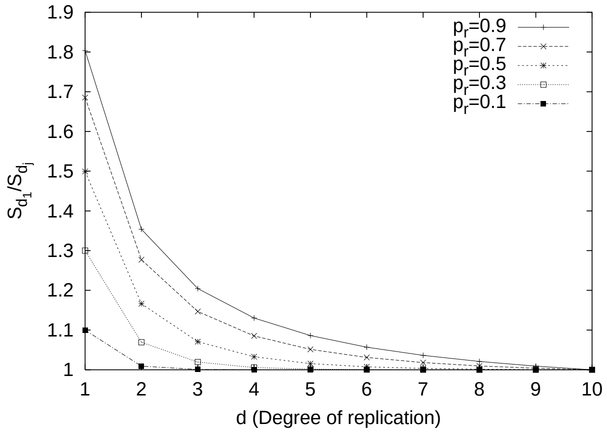
Proof: Let R_{d+k} be the mean response time when all k additional workstations are allocated to one of the programs, say J_1. Let R_{d+k_1} be the mean response time when k_1, $(0 < k_1 < k)$, workstations are allocated to J_1 and $k - k_1$ workstations are allocated to J_2. Assuming that the program is composed of one barrier, and all the tasks are of unit length, the expected response time of J_1 (or J_2) before allocation of the extra workstations is $\frac{1}{(1-p_r^d)^N}$. Therefore we have

$$R_{d+k} = \frac{1}{2}\left(\frac{1}{(1-p_r^d)^N} + \frac{1}{(1-p_r^d)^{N-k}(1-p_r^{d+1})^k}\right)$$

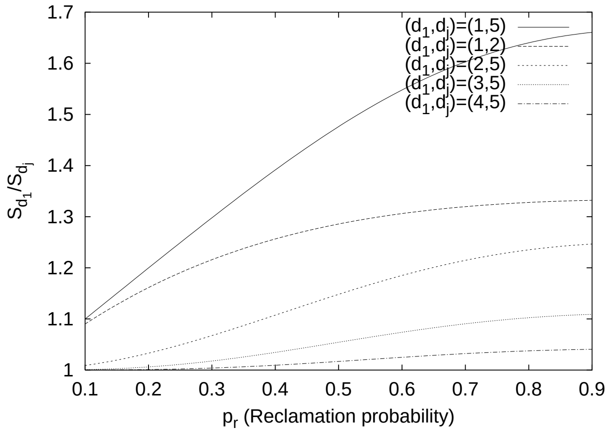
and

$$R_{d+k_1} = \frac{1}{2}\left(\frac{1}{(1-p_r^d)^{N-k_1}(1-p_r^{d+1})^{k_1}}\right.$$

$$\left.+\frac{1}{(1-p_r^d)^{N-k+k_1}(1-p_r^{d+1})^{k-k_1}}\right)$$



(a)



(b)

Fig. 4: Speedup Ratio (Allocate-to-lower-d over Allocate-to-higher-d)

Since $d \geq 1$, we know $\frac{1-p_r^{d+1}}{1-p_r^d} \geq 1$. As $k \geq 1$ and $0 < k_1 < k$, applying Lemma 5.3 we get

$$1 + (\frac{1-p_r^{d+1}}{1-p_r^d})^k \geq (\frac{1-p_r^{d+1}}{1-p_r^d})^{k-k_1} + (\frac{1-p_r^{d+1}}{1-p_r^d})^{k_1}$$

$$\frac{1}{(1-p_r^d)^{N-k}(1-p_r^{d+1})^k}[1 + \frac{(1-p_r^{d+1})^k}{(1-p_r^d)^k}] \geq$$

$$\frac{1}{(1-p_r^d)^{N-k}(1-p_r^{d+1})^k}[\frac{(1-p_r^{d+1})^{k-k_1}}{(1-p_r^d)^{k-k_1}} + \frac{(1-p_r^{d+1})^{k_1}}{(1-p_r^d)^{k_1}}]$$

$$\frac{1}{(1-p_r^d)^{N-k}(1-p_r^{d+1})^k}[\frac{1}{(1-p_r^d)^k(1-p_r^{d+1})^{-k}} + 1] \geq$$

$$\frac{1}{(1-p_r^d)^{N-k}(1-p_r^{d+1})^k}[\frac{1}{(1-p_r^d)^{k-k_1}(1-p_r^{d+1})^{-k+k_1}} + \frac{1}{(1-p_r^d)^{k_1}(1-p_r^{d+1})^{-k_1}}]$$

$$\frac{1}{(1-p_r^d)^N} + \frac{1}{(1-p_r^d)^{N-k}(1-p_r^{d+1})^k} \geq$$

$$\frac{1}{(1-p_r^d)^{N-k_1}(1-p_r^{d+1})^{k_1}} + \frac{1}{(1-p_r^d)^{N-k+k_1}(1-p_r^{d+1})^{k-k_1}}$$

Hence $R_{d+k} \geq R_{d+k_1}$, so we get a better mean response time by splitting the extra workstations among the two programs than by giving them all to one of them. ■

Theorem 3.4: The mean response time of two programs is minimized when the extra workstations are split equally among the two programs.

Proof: Now, for a fixed k, we find k_1 so as to minimize the mean response time. Note, we can minimize the mean response time, by minimizing

$$(\frac{1-p_r^{d+1}}{1-p_r^d})^{k-k_1} + (\frac{1-p_r^{d+1}}{1-p_r^d})^{k_1}$$

Let $a = \frac{1-p_r^{d+1}}{1-p_r^d}$, and let $f(x) = a^{k-x} + a^x$.

$$f'(x) = \log a \ (a^x - a^{k-x})$$

$f'(x) = 0$ when $a^x = a^{k-x}$ which is true when $x = k/2$.

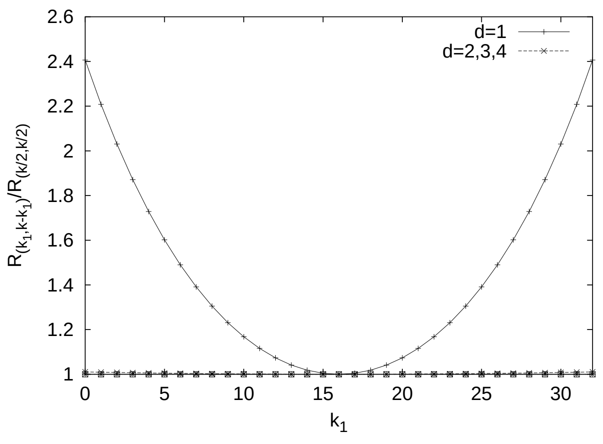
$$f''(x) = \log^2 a \ (a^x + a^{k-x})$$

$f''(k/2) > 0$, therefore $x = k/2$ is the minima of $f(x)$.

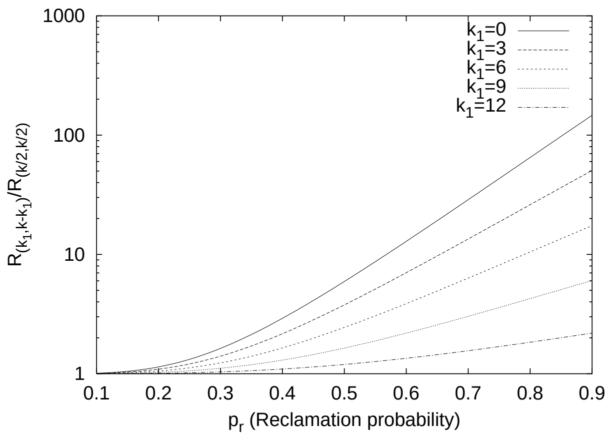
Hence, we get the best mean response time when we equally share the additional workstations among the two programs J_1 and J_2. ■

In Fig. 5 we plot the effect of distributing k extra workstations among two identical programs on their mean job response times. Unless varied, we assume $N = 32, p_r = 0.1, d = 2, k = 32$. A distribution (k_i, k_j) means, without loss of generality, k_i workstations are allocated to the first program and k_j workstations are allocated to the second program.

In both the graphs, we plot the ratio of the mean job response time for the distribution (k_1, k) over the response time for the distribution $(k/2, k/2)$. We use different values of k_1 for the plots.



(a) $p_r = 0.1,\ d = 2,\ N = 32,\ k = 32$



(b) $d = 2,\ N = 32,\ k = 32$

Fig. 5: Mean Response Time Ratio

In Fig. 5(a), we vary k_1 (x-axis) in the range 0 through $k = N = 32$. We plot the curves corresponding to d values of 1, 2, 3 and 4. In each case, the plot is symmetric about the line $x = k/2$ and the minimum mean job response time is reached for $(k/2, k/2)$. The ratio is higher for lower values of d. So it is especially important to divide the extra workstations equally among the two programs for low initial degree of replication. As seen earlier, increasing replication for higher values of d does not improve peroformance significantly. Thus the benefit of distributing the workstations among the programs is low for d = 2, 3, 4. In the plot the difference between d = 2, 3 and 4 is not discernible.

In Fig. 5(b), we vary p_r along the x-axis and plot the response time ratio from top to bottom for k_1 values of 0, 3, 6, 9 and 12. The y-axis has a logarithmic scale. A higher owner interference causes the ratio to increase significantly. For high values of p_r, even a slight imbalance in the allocation of the workstations among the two programs has a stiff penalty in terms of mean job response time.

B. Loosely Coupled Barrier

Here we consider barrier synchronized programs whose tasks do not have to be co-scheduled. So a job makes a barrier so long as all the tasks in the set are completed irrespective of whether they were all running simultaneously or not. In this master-worker scenario, the time a program needs to reach a barrier is the maximum of the time needed for each task of the corresponding set to complete. When a task is replicated, it is sufficient for one of the replicas to finish. Thus we only need to consider the replica that has the minimum completion time. Hence for a program with replicated tasks, the time needed to complete a set of tasks (reach a barrier) is the maximum of the time required to complete each of the individual tasks, which is in turn the minimum of the completion times of the task's replicas.

The probability that a task gets completed follows a geometric distribution with parameter $(1-p_r)$. The c.d.f of the distribution is given by $F(t) = 1-(1-(1-p_r))^t = 1-p_r^t$. $F^N(t) = (1-p_r^t)^N$. The mean of the maximum of N geometric variables with parameter $1-p_r$ [16] is $\mu_{N:N} = \sum_{t=0}^{\infty}(1-F^N(t)) = \sum_{t=0}^{\infty}(1-(1-p_r^t)^N)$.

Now consider the case where each task is replicated d times. The time needed to complete a task is the minimum of d geometric variables with the parameter $1-p_r$. The minimum of d geometric variables (with parameter $1-p_r$) follows a geometric distribution with parameter $1-(1-(1-p_r))^d = 1-p_r^d$. The c.d.f of the geometric distribution with parameter $1-p_r^d$ is given by $F_d(t) = 1-(1-(1-p_r^d))^t = 1-p_r^{dt}$. The mean of the maximum of N such geometric variables (with parameter $1-p_r^d$) is given by $\mu'_{N:N} = \sum_{t=0}^{\infty}(1-F_d^N(t)) = \sum_{t=0}^{\infty}(1-(1-p_r^{dt})^N)$.

$\mu'_{N:N}$ gives the mean time required to complete one barrier. If a program has B such barriers, then the time needed to complete the program is $B\mu'_{N:N}$. Similarly the time needed to complete a program with no replication is $B\mu_{N:N}$.

The completion time of the sequential version of the program is BN. Hence the speedup of the program without replication is $S_1 = \frac{BN}{B\mu_{N:N}} = \frac{N}{\sum_{t=0}^{\infty}(1-(1-p_r^t)^N)}$ and the speedup of the program with d replicas is $S_d =$

$\frac{BN}{B\mu'_{N:N}} = \frac{N}{\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^N)}$. The ratio of the speedup with replication to the speedup without replication is given by

$$\frac{S_d}{S_1} = \frac{N/(\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^N))}{N/(\sum_{t=0}^{\infty}(1-(1-p_r^{t})^N))}$$

$$\frac{S_d}{S_1} = \frac{\sum_{t=0}^{\infty}(1-(1-p_r^{t})^N))}{\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^N))}$$

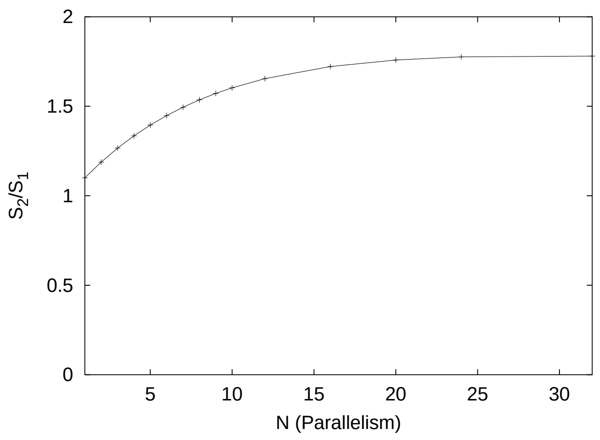
Since $0 < p_r < 1$ and $d \geq 1$, we have $p_r^d \leq p_r$. Given $t \geq 0$, $p_r^{dt} \leq p_r^t$. $(1-p_r^{dt}) \geq (1-p_r^t)$ and $(1-p_r^{dt})^N \geq (1-p_r^t)^N$, where $N \geq 1$. Hence, $(1-(1-p_r^{dt})^N) \leq (1-(1-p_r^t)^N)$, $\forall t \geq 0$. Therefore $S_d \geq S_1$. Note for $d > 1$, $S_d > S_1$.

From the above we conclude that replication always results in better speedups when $0 < p_r < 1$.

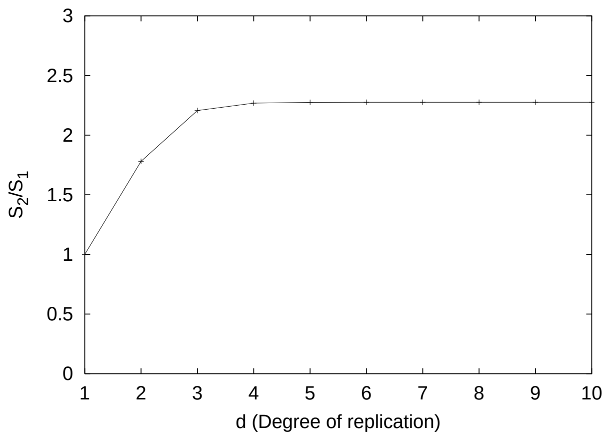
In both the graphs of Fig. 6 we plot, on the y-axis, the ratio of speedup with replication over the speedup without replication. Unless varied, we assume $N = 32$, $p_r = 0.1$ and $d = 2$.

In Fig. 6(a), we plot the speedup ratio as we vary the parallelism N on the x-axis. The speedup ratio is greater than 1, as replication provides at least as much speedup as no replication. The speedup improvement is better for larger jobs (higher values of N) than for smaller jobs. The graph tends to level off for larger N, because for larger N values, the chances of one of the workstations being reclaimed are much higher, so adding just one degree of replication is not as effective (as compared to smaller jobs).

In Fig. 6(b), we plot the speedup ratio as we vary the degree of replication d. The speedup improvement increases significantly until about $d = 3$ (for $N = 32$, $p_r = 0.1$), after which the improvement levels off.



(a) $p_r = 0.1$, $d = 2$



(b) $p_r = 0.1$, $N = 32$

Fig. 6: Speedup Ratio (Replication over No-Replication)

1) Allocating extra workstations to the same program with equal replication: Now consider the problem of allocating $1 \leq k \leq N$ additional workstations to a program whose tasks are all replicated d times.

Lemma 5.2: If $0 < p < 1$, $t > 0$, $d > 0$, and $n \geq 1$, then $\forall k$, k,n integers and $1 \leq k \leq n$, we have

$$(1-p^{dt})^{n-k}(1-p^{(d+1)t})^k \geq (1-p^{dt})^{n-1}(1-p^{(d+k)t})$$

Theorem 3.5: Replicating k tasks one additional time gives better speedup (and response time) than replicating one of the tasks k additional times.

Proof: Let S_{d+k}, S_{d+1} denote the speedups when one task is replicated k additional times ($d+k$ times) and when k tasks are replicated one additional time ($d+1$ times) respectively. Thus we have

$$S_{d+k} = \frac{N}{\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-1}(1-p_r^{(d+k)t}))}$$

$$S_{d+1} = \frac{N}{\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k)}$$

$$\frac{S_{d+1}}{S_{d+k}} = \frac{\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-1}(1-p_r^{(d+k)t}))}{\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k)}$$

From Lemma 5.2 we know that $\forall t \geq 0$, $(1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k \geq (1-p_r^{dt})^{N-1}(1-p_r^{(d+k)t})$.

Therefore, $1-(1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k \leq 1-(1-p_r^{dt})^{N-1}(1-p_r^{(d+k)t})$ which implies

$$\frac{S_{d+1}}{S_{d+k}} \geq 1$$

Thus it is better to replicate k tasks one additional time than to use all the k workstations to replicate just one proess k times more. ■

2) Allocating extra workstations to the same program with different replication: Suppose we have a program that has one task, T_1 with m total replicas and another task, T_2 with n total replicas, where $m > n$. Since the change in speedup of the program, after allocating $k \geq 1$ additional workstations to one of these tasks, only depends on the change in speedups due to the additional replication of one of these tasks, we can assume, without loss of generality, that the program only has 2 tasks T_1, T_2.

Lemma 5.1: If $x \geq 1$, $m \geq 0$, $n \geq 0$ and wlog $m \geq n$, then $1 + x^{m+n} \geq x^m + x^n$.

Theorem 3.6: Allocating extra workstations to the task that is replicated the least results in better speedup (and response time) than allocating extra workstations to other tasks.

Proof: The speedup of the program when the k extra workstations are allocated to T_1 is

$$S_{m+k} = \frac{N}{\sum_{t=0}^{\infty}(1-(1-p_r^{(m+k)t})(1-p_r^{nt}))}$$

and the speedup of the program when the k workstations are used to replicate T_2 is

$$S_{n+k} = \frac{N}{\sum_{t=0}^{\infty}(1-(1-p_r^{mt})(1-p_r^{(n+k)t}))}$$

where, $N = 2$.

From Lemma 5.1, we know $\forall t \geq 0$, $(1-p_r^{mt})(1-p_r^{(n+k)t}) \geq (1-p_r^{(m+k)t})(1-p_r^{nt})$. Hence, $(1-(1-p_r^{mt})(1-p_r^{(n+k)t})) \leq (1-(1-p_r^{(m+k)t})(1-p_r^{nt}))$. Therefore, $S_{n+k} \geq S_{m+k}$. So allocating the k additional workstations to the task which is replicated fewer times (before the allocation) is better. ■

3) Allocating extra workstations to identical programs: Consider two programs J_1 and J_2 which have $N \geq k \geq 1$ tasks each. The tasks of both the programs are replicated a total of d times each. Now we wish to allocate k additional workstations to the two programs so as to minimize the mean response time of the programs. We claim that allocating all k workstations to just one of the two programs is at least as good (better in most cases) as allocating some to each program.

Lemma 5.3: If $x \geq 1$, $m \geq 0$, $n \geq 0$ and wlog $m \geq n$, then $1 + x^{m+n} \geq x^m + x^n$.

Theorem 3.7: The mean response time is lower when all the extra workstations are allocated to either one of the programs, rather than split among both the programs.

Proof: Let R_{d+k} be the mean response time of the programs when k workstations are all allocated to one of the programs and none to the other. Let R_{d+k_1} be the mean response time of the programs when k_1, $1 \leq k_1 < k$ workstations are allocated to one of the programs and $k-k_1$ workstations are allocated to the other program. Here we assume the (identical) programs both have one set of tasks (barrier).

$$\begin{aligned}
R_{d+k} &= \frac{1}{2}[\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^N) \\
&\quad + \sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k)] \\
R_{d+k_1} &= \frac{1}{2}[\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-k_1}(1-p_r^{(d+1)t})^{k_1}) \\
&\quad + \sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-k+k_1}(1-p_r^{(d+1)t})^{k-k_1})]
\end{aligned}$$

For $d \geq 1$, $t \geq 0$ we have $(1-p_r^{(d+1)t}) \geq (1-p_r^{dt})$. Therefore, $\frac{(1-p_r^{(d+1)t})}{(1-p_r^{dt})} \geq 1$. By Lemma 5.3, we have

$$\begin{aligned}
1 + (\frac{(1-p_r^{(d+1)t})}{(1-p_r^{dt})})^k \geq (\frac{(1-p_r^{(d+1)t})}{(1-p_r^{dt})})^{k_1} \\
+ (\frac{(1-p_r^{(d+1)t})}{(1-p_r^{dt})})^{k-k_1}
\end{aligned}$$

$$\begin{aligned}
1 + \frac{(1-p_r^{(d+1)t})^k}{(1-p_r^{dt})^k} \geq \frac{(1-p_r^{(d+1)t})^{k_1}}{(1-p_r^{dt})^{k_1}} \\
+ \frac{(1-p_r^{(d+1)t})^{k-k_1}}{(1-p_r^{dt})^{k-k_1}}
\end{aligned}$$

$$\begin{aligned}
(1-p_r^{dt})^k + (1-p_r^{(d+1)t})^k \geq (1-p_r^{dt})^{k-k_1}(1-p_r^{(d+1)t})^{k_1} \\
+ (1-p_r^{dt})^{k_1}(1-p_r^{(d+1)t})^{k-k_1}
\end{aligned}$$

Multiplying both sides by $(1-p_r^{dt})^{N-k}$, we get

$$\begin{aligned}(1-p_r^{dt})^N + (1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k &\geq \\ (1-p_r^{dt})^{N-k_1}(1-p_r^{(d+1)t})^{k_1} \\ + (1-p_r^{dt})^{N-k+k_1}(1-p_r^{(d+1)t})^{k-k_1}\end{aligned}$$

$$\begin{aligned}-(1-p_r^{dt})^N - (1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k &\leq \\ -(1-p_r^{dt})^{N-k_1}(1-p_r^{(d+1)t})^{k_1} \\ -(1-p_r^{dt})^{N-k+k_1}(1-p_r^{(d+1)t})^{k-k_1}\end{aligned}$$

$$\begin{aligned}2-(1-p_r^{dt})^N - (1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k &\leq \\ 2-(1-p_r^{dt})^{N-k_1}(1-p_r^{(d+1)t})^{k_1} \\ -(1-p_r^{dt})^{N-k+k_1}(1-p_r^{(d+1)t})^{k-k_1}\end{aligned}$$

$$\begin{aligned}&(1-(1-p_r^{dt})^N) \\ &+ (1-(1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k) \leq \\ &(1-(1-p_r^{dt})^{N-k_1}(1-p_r^{(d+1)t})^{k_1}) \\ &+ (1-(1-p_r^{dt})^{N-k+k_1}(1-p_r^{(d+1)t})^{k-k_1})\end{aligned}$$

$$\begin{aligned}&\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^N) \\ &+ \sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-k}(1-p_r^{(d+1)t})^k) \leq \\ &\sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-k_1}(1-p_r^{(d+1)t})^{k_1}) \\ &+ \sum_{t=0}^{\infty}(1-(1-p_r^{dt})^{N-k+k_1}(1-p_r^{(d+1)t})^{k-k_1})\end{aligned}$$

Thus $R_{d+k} \leq R_{d+k_1}$, so we get a better mean response time if we allocate all k workstations to one of J_1 or J_2. ■

IV. Replication vs Parallelization

A. Tightly Coupled Barrier

Suppose we have a program J_1 with a sequential fraction f. Assume there is no upper bound on the maximum parallelism. By Amdahl's Law, we know that the speedup of this program has an upper bound of $S(N) = 1/(f + (1-f)/N)$, where N is the number of tasks of J_1. We shall assume that this speedup is achieved by the program when run on a set of N dedicated workstations, even though in a real scenario, the speedup achieved is much lower due to other constraints. Let us further assume that each of the N tasks has been replicated d times. Thus we are running J_1 on dN workstations. If we have $dN+1$ workstations available to us, we need to find out if it is more profitable (in terms of speedup) to increase the parallelism of J_1 to $N+1$ or to replicate an existing task one additional time.

Let S_d denote the speedup of J_1 when run on a SNOW with increased parallelism of $P+1$, and let S_{d+1} denote the speedup of J_1 when run on a SNOW with parallelism N but with extra replication. Note, for simplicity we assume p_r remains constant when the parallelism is increased to $N+1$. We have $S_d = S(N+1)(1-p_r^d)^N(1-p_r)$ and $S_{d+1} = S(N)(1-p_r^d)^{N-1}(1-p_r^{d+1})$. $S^{d+1} > S_d$ when : $S(N)(1-p_r^{d+1}) > S(N+1)(1-p_r^d)(1-p_r)$. The gain in improvement when the parallelism is increased by 1 is given by S^{d+1}/S_d.

$$\begin{aligned}\frac{S^{d+1}}{S_d} &= \frac{S(N)(1-p_r^d)^{N-1}(1-p_r^{d+1})}{S(N+1)(1-p_r^d)^N(1-p_r)} \\ &= \frac{S(N)(1-p_r^{d+1})}{(1-p_r^d)(1-p_r)} \\ &= a\frac{S(N)}{S(N+1)}\end{aligned}$$

where $a = \frac{(1-p_r^{d+1})}{(1-p_r^d)(1-p_r)}$. Note $a > 1$ for $d > 0$.

$$\begin{aligned}\frac{S^{d+1}}{S_d} &= a\frac{f+(1-f)/(N+1)}{f+(1-f)/N} \\ &= a\frac{(Nf+f+1-f)N}{(Nf+1-f)(N+1)} \\ &= a\frac{N+N^2f}{(N+1)+(N^2-1)f}\end{aligned}$$

$$\frac{S^{d+1}}{S_d} - 1 = b[a(N+N^2f) - (N+1) - (N^2-1)f]$$

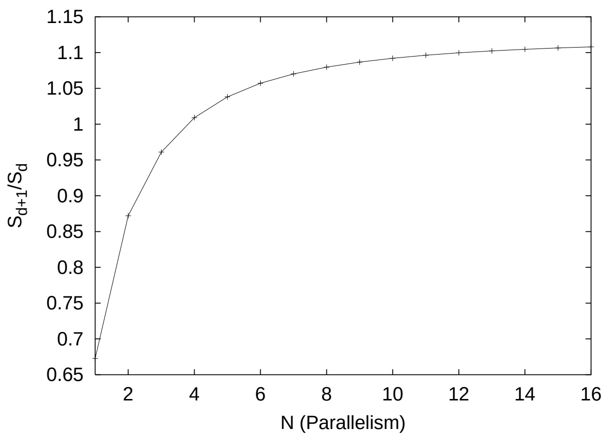
where $b = 1/[N+1+(N^2-1)f]$. Note $b > 0$ for $N > 0$.

$$\begin{aligned}\frac{S^{d+1}}{S_d} - 1 &= b[aN + aN^2f - N - N^2f - (1-f)] \\ &= b[(a-1)fN^2 + (a-1)N - (1-f)] \\ &= (a-1)b[fN^2 + N - \frac{1-f}{a-1}]\end{aligned}$$

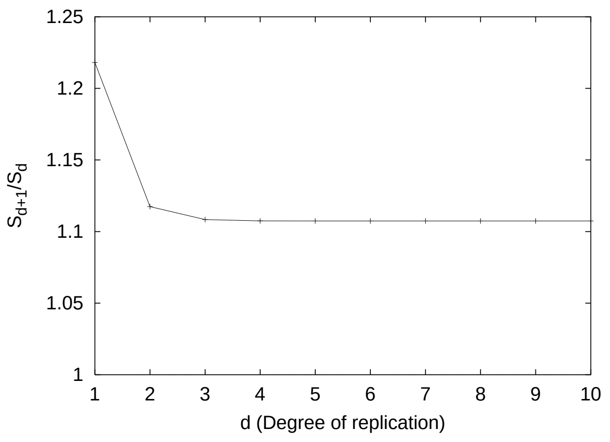
1) Example: N=1: If $N = 1$,

$$\frac{S^{d+1}}{S_d} - 1 = (a-1)b[f + 1 - \frac{1-f}{a-1}] = b(af + a - 2)$$

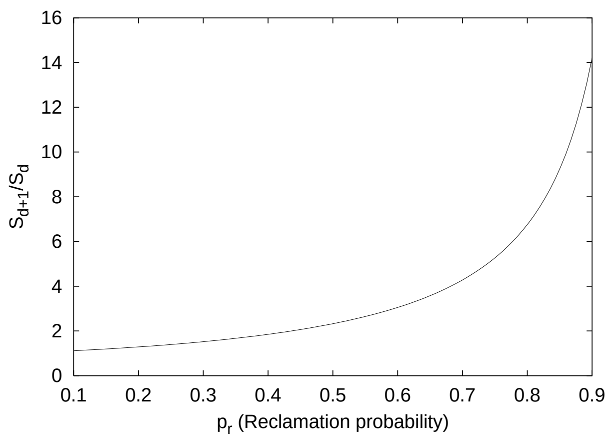
Thus it is always better to increase replication (rather than parallelism) if $f > 2/a - 1$.



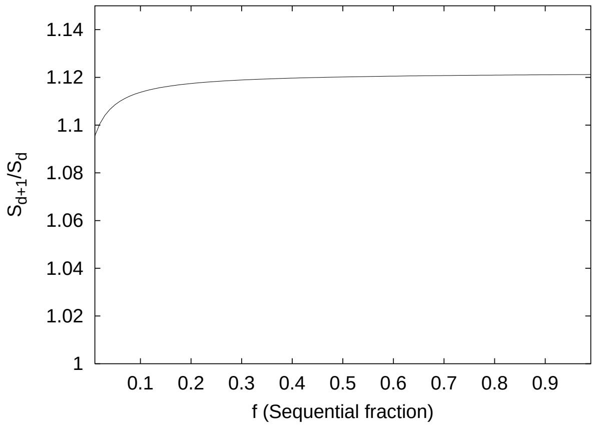
(a) $p_r = 0.1,\ d = 2,\ f = 0.2$



(b) $p_r = 0.1,\ N = 32,\ f = 0.2$



(c) $d = 2,\ N = 32,\ f = 0.2$



(d) $p_r = 0.1,\ d = 2,\ N = 32$

Fig. 7: Speedup Ratio (Replication over Parallelism)

In Fig. 7 we plot the ratio of speedup with increased replication over the speedup with increased parallelism. Unless varied, we assume $d = 2$, $N = 32$, $p_r = 0.1$ and the sequential fraction of the parallel program $f = 0.2$.

In Fig. 7(a), we vary N along the x-axis and study its effect on the ratio of speedup with increased replication over speedup with increased parallelism. For this choice of parameters it is better to increase replication for all $N \geq 4$. For $N < 4$, increasing parallelism gives a better performance. When N is large, using just one extra workstation to replicate a task has a reduced effect on overall performance. Thus we see the ratio levelling off.

In Fig. 7(b), we vary the initial degree of replication d (before using the extra workstation). An increase in the initial degree of replication, means the amount of improvement possible, by replicating one of the tasks once more, is lower. Hence the improvement by increased replication is less relative to the improvement possible by increasing parallelism. So we see a drop in the speedup ratio. For higher values of d, using one extra replication of one task of the program has a low effect on overall speedup and thus we see the speedup ratio levelling off.

In Fig. 7(c), we vary p_r. Replication is especially helpful when owner interference is high. Thus when p_r increases, the speedup ratio also increases significantly.

In Fig. 7(d), we vary the sequential fraction of the program f. We notice that the sequential fraction of the program has a very low effect on the speedup ratio.

V. Proofs

Lemma 5.1: If $0 < p < 1$, $n > 0$ and $k > 0$, then $\forall m > n$, we have

$$\frac{1-p^{n+k}}{1-p^n} > \frac{1-p^{m+k}}{1-p^m}$$

Proof: Since $k > 0$, $n > 0$ and $0 < p < 1$, $p^n > p^{n+k}$. And also since $m > n$, $p^{m-n} < 1$ which implies $(1-p^{m-n}) > 0$. Therefore, we have

$$p^n(1-p^{m-n}) > p^{n+k}(1-p^{m-n})$$

$$p^n - p^m > p^{n+k} - p^{m+k}$$

$$-p^m - p^{n+k} > -p^n - p^{m+k}$$

$$1-p^{n+k}+p^{m+n+k}-p^m > 1-p^{m+k}+p^{m+n+k}-p^n$$

$$1-p^{n+k}+p^m(p^{n+k}-1) > 1-p^{m+k}+p^n(p^{m+k}-1)$$

$$(1-p^{n+k})(1-p^m) > (1-p^{m+k})(1-p^n)$$

$$\frac{1-p^{n+k}}{1-p^n} > \frac{1-p^{m+k}}{1-p^m}$$

■

Lemma 5.2: If $0 < p < 1$, $t > 0$, $d > 0$, and $n \geq 1$, then $\forall k$, k,n integers and $1 \leq k \leq n$, we have

$$(1-p^{dt})^{n-k}(1-p^{(d+1)t})^k \geq (1-p^{dt})^{n-1}(1-p^{(d+k)t})$$

Proof: Since $d > 0$ and $t > 0$, from Lemma 5.1 we have for $1 \leq i \leq k-1$

$$\frac{1-p^{dt+t}}{1-p^{dt}} \geq \frac{1-p^{(d+i)t+t}}{1-p^{(d+i)t}}$$

Therefore,

$$\frac{(1-p^{dt+t})^{k-1}}{(1-p^{dt})^{k-1}} \geq \prod_{i=1}^{k-1} \frac{1-p^{(d+i)t+t}}{1-p^{(d+i)t}}$$

$$(1-p^{(d+1)t})^{k-1} \geq (1-p^{dt})^{k-1} \prod_{i=1}^{k-1} \frac{1-p^{(d+i+1)t}}{1-p^{(d+i)t}}$$

$$\geq (1-p^{dt})^{k-1} \frac{1-p^{(d+k)t}}{1-p^{(d+1)t}}$$

$$(1-p^{(d+1)t})^k \geq (1-p^{dt})^{k-1}(1-p^{(d+k)t})$$

$$(1-p^{dt})^{n-k}(1-p^{(d+1)t})^k \geq (1-p^{dt})^{n-1}(1-p^{(d+k)t})$$

■

Lemma 5.3: If $x \geq 1$, $m \geq 0$, $n \geq 0$ and wlog $m \geq n$, then $1 + x^{m+n} \geq x^m + x^n$.

Proof:

Case 1: $n = 0$

$n = 0$ implies $x^{m+n} = x^m$ and $x^n = 1$. So $1 + x^{m+n} = 1 + x^m = x^m + x^n$. Thus, $1 + x^{m+n} \geq x^m + x^n$.

Case 2: $n > 0$

Let $f(x) = 1 + x^{m+n} - x^m - x^n$. Now, $f'(x) = (m+n)x^{m+n-1} - mx^{m-1} - nx^{n-1}$.

$$f'(x) = (m+n)x^{m+n-1}[1 - \frac{m}{m+n}\frac{1}{x^n} - \frac{n}{m+n}\frac{1}{x^m}]$$
$$= (m+n)x^{m+n-1}$$
$$[1 - \frac{1}{x^n}(\frac{m}{m+n} + \frac{n}{m+n}\frac{1}{x^{m-n}})]$$

Since, $x \geq 1$ and $m \geq n$, $\frac{1}{x^{m-n}} \leq 1$,

$$(\frac{m}{m+n} + \frac{n}{m+n}\frac{1}{x^{m-n}}) \leq 1$$

Because $n > 0$, $\frac{1}{x^n} \leq 1$, therefore $f'(x) \geq 0$ so $f(x)$ is increasing when $x \geq 1$. At $x = 1$, $f(1) = 0$ which implies $f(x) \geq 0$, for $x \geq 1$. Hence, $1 + x^{m+n} \geq x^m + x^n$. ■

VI. Conclusions

We have analyzed the performance improvements resulting from task replication of batch parallel programs running on a SNOW. Specifically, we have derived formulas to calculate the speedup and efficiency improvements due to task replication. With our analysis we have shown that replicating tasks of parallel programs can result in significant speedup improvements. Also, for some workloads, replication can also improve efficiency. Furthermore, when the probability of workstation reclamation rises, the speedup and efficiency improvements due to replication increase. Likewise, as job parallelism increases, replication becomes even more beneficial in improving speedup.

We have also analyzed the problem of using extra workstations to replicate tasks of a parallel program and show how to distribute the extra workstations among the tasks. Specifically, for the workload models considered, if we have k extra workstations, we have shown it is better to replicate k tasks one additional time than to replicate one of the tasks k additional times. If there is only one extra workstation, we have shown it is best to allocate the extra workstation to the least replicated task. Finally, if we have extra workstations to distribute among two identical programs, distributing the workstations equally between the two programs gives least mean response time for tightly coupled workload and giving all the extra workstations to one of the programs gives least mean response time for loosely coupled workload.

Lastly, we have presented an analysis of the trade-off between using an extra workstation to increase parallelism and increasing replication, and have found that replication can be more beneficial than increasing parallelism for a range of tightly-coupled workloads.

We have made a strong case for considering the use of replication in the design and implementation of scheduling policies for SNOWs.

We plan on further investigating the speedup improvements of replication for the master-worker workload and for workloads with un-equal task demands. We also plan to consider the problem of distributing extra workstations among several batch parallel programs.

REFERENCES

[1] A. Acharya, G. Edjlali, and J. Saltz, "The utility of exploiting idle workstations for parallel computation," in *Proc. 1997 ACM SIGMETRICS*, 1997, pp. 225–236.

[2] M. Adlet, Y. Gong, and A. Rosenberg, "Optimal sharing of bags of tasks in heterogeneous clusters," in *Proc. of the fifteenth annual ACM Symposium on Parallel Algorithms and Architectures*, 2003, pp. 1–10.

[3] T. Anderson, D. Culler, and D. Patterson, "A case for networks of workstations: Now," *IEEE Micro*, Feb 1995.

[4] R. Arpaci, A. Dusseau, A. Vahdat, L. Liu, T. Anderson, and D. Patterson, "The interaction of parallel and sequential workloads on a network of workstations," in *Proc. 1995 ACM SIGMETRICS*, 1995.

[5] S. Cho, "Competitive execution in a distributed environment," Ph.D. dissertation, University of California, Los Angeles, 1996.

[6] E. Heymann, M. Senar, E. Luque, and M. Livny, "Evaluation of strategies to reduce the impact of machine reclaim in cycle-stealing environments," in *Proc. First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001, pp. 320–328.

[7] S. Leutenegger and X. Sun, "Limitations of cycle stealing for parallel processing on a network of homogeneous workstations," *Journal of Parallel and Distributed Computing*, vol. 43, no. 3, pp. 169–178, 1997.

[8] J. Pruyne and M. Livny, "Interfacing condor and pvm to harness the cycles of workstation clusters," *Journal on Future Generations of Computer Systems*, vol. 12, 1996.

[9] ——, "Managing checkpoints for parallel programs," in *Workshop on Job Scheduling Strategies for Parallel Processing, IPPS 96*, 1996.

[10] A. L. Rosenberg, "Optimal schedules for cycle-stealing in a network of workstations with a bag-of-tasks workload," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 2, pp. 179–191, February 2002.

[11] T. Sterling, D. Becker, D. Savarese, J. Dorband, U. Ranawake, and C. Packer, "Beowulf: A parallel workstation for scientific computation," in *Proc. of the International Conf. on Parallel Processing*, 1995.

[12] M. Litzkow and M. Livny, "Experience with the condor distributed batch system," in *Proc. of IEEE Workshop on Experimental Distributed Systems*, Oct 1990.

[13] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - a hunter of idle workstations," in *Proc. of the 8th International Conference on Distributed Computing Systems (ICDCS)*, 1987, pp. 104–111.

[14] D. Eager, J. Zahorjan, and E. Lazowska, "Speedup versus efficiency in parallel systems," *IEEE Transactions on Computers*, vol. 38, no. 3, pp. 408–423, March 1989.

[15] J. Goux, S. Kulkarni, M. Yoder, and J. Linderoth, "An enabling framework for master-worker applications on the computational grid," in *Proc. of the 9th IEEE International Symposium on High Performance Distributed Computing*, 2000, pp. 43–50.

[16] H. David, *Order Statistics*. John Wiley and Sons, Inc., 1970.