Speedup of Sequential Program Execution on a Network of Shared Workstations

Sung-Hyun Cho and Sung-Syck Jun

Abstract

We present competition protocols to speed up the execution of sequential programs on a network of shared workstations in the background by exploiting their wasted computing capacity, without interfering with processes of workstation owners. In order to argue that competition protocols are preferable to migration protocols in this situation, we derive the closed form solutions for the speedup of competition protocols and migration protocols, and simulate both of protocols ,under comparable overhead assumptions. Based on our analytic results and simulation results, we show that competitive execution is superior to process migration, and that competitive execution can finish sequential programs significantly faster than noncompetitive execution, especially when the foreground load is sufficiently high.

I. Introduction

Networks of workstations(NOW)[1] have been increasingly prevalent in almost every computing environment, but much of computing capacity of a typical workstation has gone unused. Since the middle of the 1980's, many researchers have tried to harness idle workstations, for example, NEST[2], Benevolent Bandit[3], Stealth[4], Condor[5], Butler[6], REM[7], the V System [8], and Utopia[9]. The goal of those systems is to execute processes on remote idle workstations, maximizing the utilization of workstations.

In this paper, we consider that all processes of workstation owners are *foreground* processes regardless of their priority, and that the other processes are *background* processes. We may dedicate a subset of workstations in a NOW to execute background programs, or we may share the workstations with interactive foreground processes. These two NOW environments are called a *dedicated NOW* (DNOW) and a *shared NOW* (SNOW) respectively.

One of key issues in a SNOW is how intrusive a background program is to foreground processes and vice versa. We assume that workstation operating systems manage resources so that background programs do not interfere with foreground processes, but foreground processes will interfere with the performance of background programs. We refer to workstations scheduled in this way as *variable-speed* processors for background programs. When a variable-speed processor executes a foreground process, we say that it is *f-busy*. Otherwise, it is *f-idle*. The terms "workstations" and "processors" are interchangeably used, and a variable-speed processor is simply called a processor in case of no confusion.

The progress rate of processes on a DNOW is constant, but that on a SNOW may unpredictably vary due to processor sharing. This unpredictable variability may drastically degrade the performance of background programs. In this paper, we present competition protocols[10] to overcome this problem. Assuming that the processors participating in competition are the same as those participating in migration, we compare the performance of competitive execution with that of process migration based on our analytic results and simulation results.

II. Competition Protocols

Competition protocols are *transparent* operating system facilities that involve creating multiple clones p_1, p_2, ..., p_n of a process p on n different variable-speed processors, and making them "compete". Then competition protocols monitor which clone is ahead from time to time, and determine which clone is most ahead at any given time, based on the number of instructions that clones have executed, or the CPU times clones have used. If one clone is sufficiently far ahead, competition protocols pause and propagate the state of the clone ahead to the ones that are behind. The clones that are behind use the propagated state to continue execution, that is, the clones that are behind jump forward to the state of the clone ahead without executing instructions in p themselves.

If for any reason there is variation in the progress of the clones, so that one clone is ahead at some times, but another is

---

Manuscript received March 12, 1997; accepted August 21, 1997.

S. H. Cho is with the Dept. of computer and information communication in Hongik Univertiy, Korea.

S. S. Jun is with the Dept. of computer science in Korea University, Korea.

ahead at other times, then a set of competing clones may outperform any single copy. The goal of competitive execution is to execute programs on idle processors on a SNOW without interfering with foreground processes, minimizing the completion time of programs. For competition, we require the runtime behavior of each process to be deterministic.

1. Assumptions

We assume that a SNOW consists of M processors and a variable-delay communication network connecting processors, and that all processors are physically identical. Each processor has its own memory, and processors communicates with each other via messages. Their execution speeds for background processes change unpredictably, and the execution speed of a processor is uncorrelated with those of other processors. In this paper, we make only one assumption about execution speeds of processors for background processes: they all have the same average execution speed. However, background execution speeds may fluctuate dynamically on processors over time.

For a sequential process *sp,* we assume that a competition protocol replicates *sp* and statically assign k clones sp_1, sp_2, ..., sp_k to k processors out of M processors where $k \leq M$. We also assume that a migration protocol migrates *sp* among k such processors for fair comparison. At any given time *sp* is executed by one of them, which is called a *hosting* processor.

We assume a simple policy for migration and competition respectively: when a process is preempted due to foreground processes, a migration protocol migrates the process immediately to a f-idle processor if one is available. Otherwise, the migration protocol migrates the process later as soon as one processor becomes f-idle. This migration protocol is called a *MIG* protocol. On the other hand, a competition protocol, called a *COMP* protocol, propagates states immediately when a clone ahead is preempted. It is assumed that the delay for migration and state propagation is zero in the *MIG* protocol and the *COMP* protocol. When the delay for migration or state propagation is non-zero but constant *D*, they are called a *MIG_D* protocol and a *COMP_D* protocol respectively. However, we assume that the decision for migration and state propagation is made instantly in all the migration or competition protocols in this paper.

While migration protocols migrate a process, we assume that it cannot execute until migration is finished. Similarly when competition protocols propagate a state of a clone, they *freeze* execution of all the other clones instantaneously. Then they start to propagate its state to the other clones.

2. An Example of Competition on Three Processors

Consider an example in Fig. 1 where there are three variable-speed processors *P1*, *P2*, and *P3*. Now we want to execute a non-interactive sequential process *sp* in the background on these

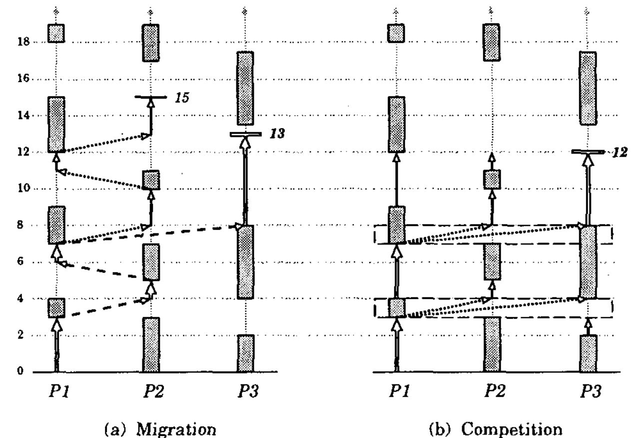


Fig. 1. Execution on Three Variable-speed Processors.

three processors. Shaded rectangles represent foreground activities, and we cannot know what the future foreground activities are going to be on these processors.

Foreground activities on *P1, P2,* and *P3* are identical in Fig. 1 (a) and (b) where numbers represent real time. The periods of freezing are shown by dashed rectangles in Fig. 1 (b), and the faster trace of execution is shown by thick vertical arrows while the slower one is shown by thin vertical arrows. We assume that the delay for process migration or state propagation is 1 unit of time, and that it takes 10 units of time to run *sp* on a dedicated processor.

The dashed arrows in Fig. 1 (a) show the optimal migration path where migration completes *sp* in 13 units of time. In order to obtain the shortest execution time, it is necessary to migrate *sp* from *P1* to *P3* rather than *P2* at real time 7. However, when migration decision is made at real time 7, it cannot be known that *P3* is a better choice than *P2* without the complete knowledge of the future foreground activities on *P2* and *P3.*

On the other hand, Fig. 1 (b) shows that competition completes *sp* in 12 units of time. Unlike migration, competition does not need to know the future foreground activities on *P2* and *P3* at real time 7 to finish *sp* in the shortest execution time. Competition easily obtains this benefit at the cost of redundant execution of clones.

Now we present other reasons why competition protoocols are preferable to migration protocols in a SNOW. Migration protocols have to choose one *target* processor out of multiple candidate processors for migration. However, the target may become f-busy before migration is finished, even though other candidates are still f-idle. When the migration delay is large or the workload of foreground processes on processors changes fast or drastically, migration may be often unsuccessful. Even if migration is successful, the f-idle period of the target may be smaller than those of the other candidates. That is, migration protocols choose any of the f-idle periods of all the candidates, including the *minimum* and the *maximum.*

On the other hand, the f-idle period in competition protocols is not a typical f-idle period but the *maximum* of f-idle periods; when more than one clone start to execute at the same time after state propagation, all of them must have made the same progress when any of them is preempted earliest. But only the clone that is preempted last can propagate its state.

3. Optimal Speedup of Competition

The completion time of a clone can be stretched due to the preemption of foreground processes. Let W be a completion time of a sequential process on a dedicated processor and let T denote a completion time of the process on a variable-speed processor. The stretching ratio SR of the variable-speed processor is defined as $SR = T/W$, and we assume that the mean stretching ratio $\overline{SR}$ does not change for the entire computation of applications.

We assume that the mean foreground processor utilization ρ is statistically identical at all processors. Then the mean background execution speed at all processors is also statistically identical. Now assume that we run a program on one processor in the background without competition, and that we run the program in the background with competition on N ($N > 1$) such processors with the same ρ. Let $\overline{T_{nc}(1,\rho)}$ and $\overline{T_c(N,\rho)}$ denote the mean completion time of the program without competition and that with competition respectively. We define the speedup of competition, $SP(1,N,\rho)$, as

$$SP(1,N,\rho) = \frac{\overline{T_{nc}(1,\rho)}}{\overline{T_c(N,\rho)}} \tag{1}$$

Let an *ideal* competition protocol make state propagation decision and state propagation itself instantaneously without any overhead at optimal times.

Theorem 1 : Assume that all processors alternate f-idle periods and f-busy periods, and that the foreground utilization of all processors is ρ. Then the optimal speedup of the ideal competition protocol on k independent processors over one processor is

$$1+\rho+\cdot\cdot\cdot+\rho^{k-1}. \tag{2}$$

The proof of this theorem is omitted here, but is found in[10]. This theorem is true for independent processors with any underlying distributions as long as they have the same ρ, and shows that the ideal competition protocol has a diminishing return in adding another processor. The best speedup from competition occurs when ρ is very high. But that's exactly the same time when it takes the longest. So we have the tradeoff; we can finish soon if ρ is low because there are lots of spare cycles, but competition probably won't help much. Or if ρ is high, there is high degree of competition efficiency and thus we can finish soon by cloning more.

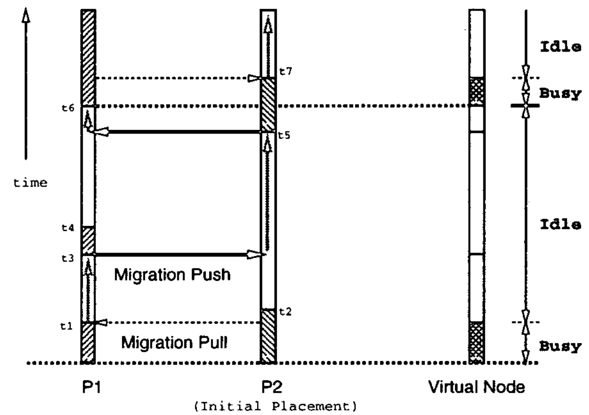


Fig. 2. MIG Protocol.

III. MIG Protocol and COMP Protocol

We assume that the lengths of f-busy periods at all processors are independent and identically distributed (i.i.d.) random variables from an exponential distribution B(t), with mean τ_b, and corresponding density b(t). Then,

$$B(t) = 1 - e^{-t/\tau_b}, \quad t \geq 0. \tag{3}$$

Likewise, the lengths of f-idle periods at all processors are i.i.d. random variables from an exponential distribution I(t), with mean τ_i, and density i(t). Then processors alternate between f-busy and f-idle states at constant rates[10], and each processor is f-busy a fraction $\rho = \tau_b / (\tau_i + \tau_b)$ of time, and f-idle the remaining fraction 1-ρ of time. The mean stretching ratio can be expressed in terms of τ_i and τ_b on a processor as follows[10];

$$\overline{SR} = 1 + \tau_b/\tau_i. \tag{4}$$

Now we derive the closed form solutions for the speedup of the *MIG* protocol and the *COMP* protocol.

1. MIG Protocol

We consider two processors *P1* and *P2* in Fig. 2 where *sp* initially starts on *P2*. Shaded periods indicate that processors are f-busy, and white periods indicate that processors are f-idle. A solid horizontal arrow indicates a process migration (called *migration push*) at the end of a f-idle period, while a dotted horizontal arrow indicates a *migration pull* at the beginning of a f-idle period. We need to distinguish a migration push and a migration pull when the migration delay is non-zero. The execution trace of *sp* is shown by thick vertical arrows in Fig. 2

In order to analyze the performance of the *MIG* protocol, we project f-busy and f-idle periods on *P1* and *P2* horizontally onto an imaginary processor, called a *virtual node*. If both *P1* and *P2* are f-busy at a given time, the virtual node is called *busy*.

Otherwise, the virtual node is called *idle* at the moment. A period that is continuously busy or idle is called a busy period or an idle period respectively. The initial f-busy periods on two processors are in fact the residual f-busy periods when *sp* starts. Since the distribution of time remaining for an exponentially distributed random variable is independent of the acquired "age" of that random variable[11], the length of each busy period on the virtual node is simply the minimum of the lengths of two f-busy periods on *P1* and *P2*.

An idle period on the virtual node may consist of more than one segment where each segment indicates that *sp* executes on a processor during that period. Busy periods and idle periods on the virtual node alternate. We analyze the performance of the *MIG* protocol on the virtual node based on the renewal period[12]. If we know the means of busy periods and idle periods on the virtual node, we can find the performance of the *MIG* protocol by using equation (4).

Let B_k and I_k denote a busy period and an idle period respectively on a virtual node when there are *k* participating processors. Then B_k is the minimum of *k* exponentially distributed random variables with the same mean τ_b, and thus

$$\overline{B_k} = \tau_b \,/\, k. \tag{5}$$

Now consider the mean of an idle period. The probability that a processor is f-busy at any given time, is ρ. If *sp* starts at *P1* initially in Fig. 2, *P1* can start to execute *sp* without a migration pull, and *P1* is called a *proper* processor. Otherwise, *P1* is called an improper processor.

Let *X* denote the number of migrations excluding the migration pull in an idle period. Then the number of segments in an idle period is *X*+1. The probability mass function of *X* is given by a modified geometric distribution[13]:

$$P_X(i) = f \cdot s^i, \quad i = 0, 1, 2, \cdots \tag{6}$$

where *s* is the probability that another processor is f-idle when *sp* is preempted at one processor, and $f = 1 - s$. Then

$$\overline{X} = s / f = s / (1 - s).$$

An idle period consists of the random sum of segments, and each segment is exponentially distributed as a f-idle period due to the memoryless property of exponential distributions. Let $\overline{I_2}$ be the mean of I_2. Then $\overline{I_2} = \tau_i(\overline{X} + 1) = \tau_i/(1-s)$. Thus the mean stretching ratio of the *MIG* protocol on two processors is given by

$$SR_{MIG}(2) = 1 + \frac{\overline{B_2}}{\overline{I_2}} = 1 + \frac{\tau_b}{\tau_i} \cdot \frac{(1-s)}{2}. \tag{7}$$

The speedup of the *MIG* protocol with two processors over a single processor is

$$SP_{MIG}(2) = \frac{1 + (\tau_b/\tau_i)}{SR_{MIG}(2)} = \frac{1 + (\tau_b/\tau_i)}{1 + (\tau_b/\tau_i) \cdot (1-s)/2}. \tag{8}$$

Since the *MIG* protocol does not have any overhead, equation (8) should be the same as the optimal speedup in Theorem. 1, which is 1 + ρ. By setting $SP_{MIG}(2) = 1 + \rho$, we have

$$s = \frac{\tau_i}{\tau_i + 2\tau_b}. \tag{9}$$

2. COMP Protocol

Consider Fig. 3 where clones sp_1 and sp_2 start to execute at *t1* on *P1* and at *t2* on *P2* respectively. A solid arrow indicates a state propagation. Note that *P1* preempts sp_1 at *t6* and propagates its state even though *P2* is f-busy.

An idle period on a virtual node ends when both of processors is f-busy. Thus we need to know what is the probability that the other processor is f-busy or f-idle at the end of a state propagation. Since state propagation is instantaneous, this probability is the same as the probability that the other processor is f-idle at the moment of preemption on a processor. Thus this probability will be the same as *s* in equation (9).

Let *Y* denote the number of state propagations in an idle period. The number of segments in an idle period of the *COMP* protocol is *Y* because state propagations occur at the end of each segment including the last one. Thus the probability mass function of *Y* is given by a geometric distribution[13]:

$$P_Y(i) = f \cdot s^i, \quad i = 1, 2, \cdots \tag{10}$$

where $s = \tau_i / (\tau_i + 2\tau_b)$ and f = 1-S. Thus

$$\overline{Y} = 1/f = 1 + \tau_i / 2\tau_b.$$

Let I_2 denote the length of an idle period when there are two participating processors. Then

$$\overline{I_2} = \tau_i \cdot \overline{Y} = \tau_i\,(1 + \tau_i/2\tau_b). \tag{11}$$

Since $\overline{B_2} = \tau_b / 2$, the mean stretching ratio of the *COMP* protocol on two processors is given by

$$SR_{COMP}(2) = 1 + \frac{\overline{B_2}}{\overline{I_2}} = \frac{(\tau_i + \tau_b)^2}{\tau_i(\tau_i + 2\tau_b)}. \tag{12}$$

The speedup of the *COMP* protocol with two processors over a single processor is

$$SP_{COMP}(2) = \frac{1 + (\tau_b/\tau_i)}{SR_{COMP}(2)} = 1 + \frac{\tau_b}{\tau_i + \tau_b} = 1 + \rho. \tag{13}$$

Thus the performance of the *COMP* protocol is the same as the optimal speedup given in Theorem 1. Our analytic results here

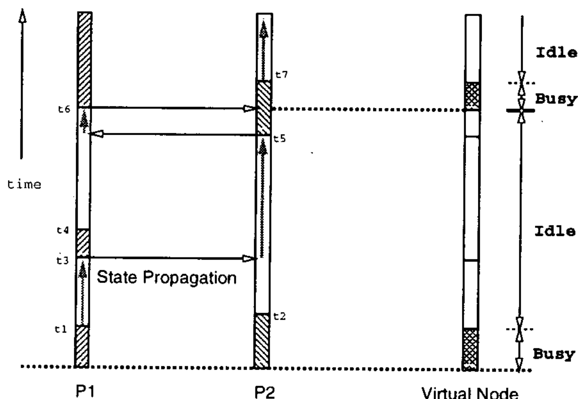


Fig. 3. COMP Protocol.

must agree with the optimal speedup because the *MIG* protocol and the *COMP* protocol never waste any available CPU cycles.

IV. MIG_D Protocol and COMP_D Protocol

Migration in the *MIG* protocol is always successful because migration is instantaneous, but migration in the *MIG_D* protocol may not be successful due to the migration delay. When migration is unsuccessful, we assume that the target processor treats *sp* in the same way as it preempts *sp*. We derive the speedup solutions of the *MIG_D* protocol and the *COMP_D* protocol.

1. MIG_D Protocol

The activities of foreground processes on *P1* and *P2* in Fig. 4 are the same as those in Fig. 2. A solid arrow indicates a migration push and a dotted arrow indicates a migration pull. An idle period in Fig. 4 consists of black periods and white periods. A black period denotes the migration delay *D*, which is also shown by a shaded rectangle. Only white periods will be effectively available to execute *sp*.

Let A_k and U_k denote the sum of white periods and that of black periods in an idle period on a virtual node respectively when the number of participating processors is *k*. A_k is called an *available* time. Since only the available time in a renewal period can be used to execute *sp*, the stretching ratio will be $1+(B_k+U_k)/A_k$.

An idle period can start with a white or a black period depending on whether *sp* is positioned at the proper processor at the beginning of an idle period. Let *X* denote the number of migrations except the migration pull in an idle period. Then the random variable *X* will be distributed as in equation (6).

If $D > \tau_i$, a f-idle period is likely to be available locally, and we are interested in the case where $D < \tau_i$. Let *I* be a random variable which represents a residual f-idle period. The probability of migration success is $P[I > D] = e^{-D/\tau_i}$ because of the memoryless property of an exponential distribution.

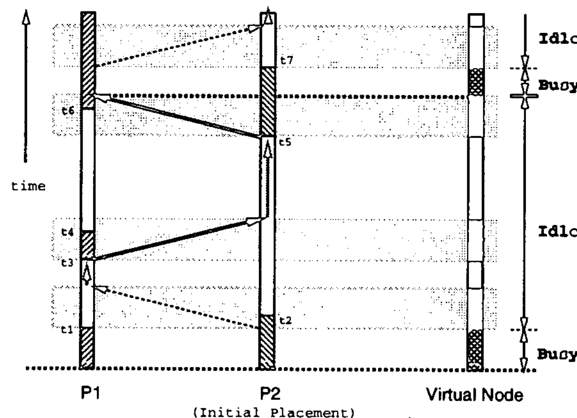


Fig. 4. MIG_D Protocol.

Let the mean of an available time in an idle period be $\overline{A_2}$. Then

$$\overline{A_2} = \frac{\tau_i}{2} + \frac{\tau_i}{2} P[I > D] + \tau_i \cdot \overline{X} \cdot P[I > D]. \tag{14}$$

The first and the second terms account for the case that *sp* is positioned at the proper processor and at the improper processor in the beginning of an idle period respectively. The last term accounts for the accumulated f-idle periods on *P1* and *P2* in an idle period. The second and third terms are multiplied by $P[I > D]$ because *sp* has the chance to run only when migration is successful. Since the mean number of migration pushes is $\overline{X} = s/(1-s) = \tau_i / 2\tau_b$,

$$\overline{A_2} = \frac{\tau_i}{2}[1 + (1 + \frac{\tau_i}{\tau_b})e^{-D/\tau_i}]. \tag{15}$$

Here $\overline{B_2} = \tau_b/2$, and let $\overline{U_2}$ be the mean of the sum of the migration delays in an idle period. Then,

$$\overline{U_2} = \frac{D}{2} + \overline{X} \cdot D = \frac{D}{2}(1 + \frac{\tau_i}{\tau_b}). \tag{16}$$

The first term accounts for the migration pull, and the second term accounts for the migration delay due to migration push regardless of its success or failure. Thus the stretching ratio of the *MIG_D* protocol on two processors is

$$SR_{MIG-D}(2) = 1 + \frac{\tau_b + \omega D}{\tau_i(1 + \omega e^{-D/\tau_i})} \tag{17}$$

where $\omega = 1 + \tau_i/\tau_b$.

2. COMP_D Protocol

The activities of foreground processes on *P1* and *P2* in Fig. 5 are the same as those in Fig. 3, and a solid arrow indicates a state

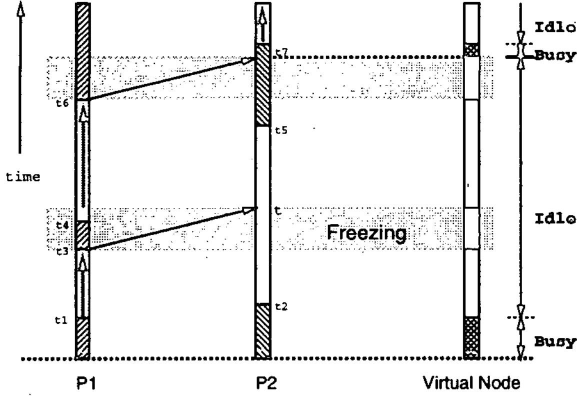


Fig. 5. COMP_D Protocol.

propagation. An idle period in Fig. 5 consists of black periods and white periods, and a black period denotes the state propagation delay D, which is also shown by a shaded rectangle. Only white periods will be effectively available to execute clones.

We say that a state propagation is successful if a state propagation is completed when one of participating processors is f-idle, and thus *sp* can make its progress without any interruption except the propagation delay. This occurs when the f-busy period at the state propagator is smaller than D or state propagation is completed when the other processor is f-idle. We approximately use $s = \tau_i / (\tau_i + 2\tau_b)$ in equation (9) as the probability of the latter case. Then the probability that a state propagation succeeds is given by

$$s_1 = 1 - e^{-D/\tau_b} + e^{-D/\tau_b} \cdot \frac{\tau_i}{\tau_i + 2\tau_b}, \tag{18}$$

and the probability that state propagation fails is $f_1 = 1 - s_1$. Let Y be the number of state propagations in an idle period. Then the probability mass function of Y is given by the geometric distribution in equation (10) where $s = s_1$ and $f = f_1$. Thus

$$\overline{Y} = 1/f_1 = (1 + \frac{\tau_i}{2\tau_b})e^{D/\tau_b}. \tag{19}$$

When a state propagation is completed, two clones may start execution at the same time, but only the clone with the longer f-idle period can propagate its state. Thus the corresponding segment here is not a typical f-idle period, but the maximum of two f-idle periods. When one clone starts earlier than the other, the corresponding segment is a typical f-idle period.

Two clones start at the same time when the f-busy period on the state propagator is smaller than D and the other processor is f-idle when the state propagation is completed. This probability is

$$b = (1 - e^{-D/\tau_b})\frac{\tau_i}{\tau_i + 2\tau_b}, \tag{20}$$

and the probability that one clone starts earlier than the other is

$$c = e^{-D/\tau_b}\frac{\tau_i}{\tau_i + 2\tau_b} + (1 - e^{-D/\tau_b})\frac{2\tau_b}{\tau_i + 2\tau_b}. \tag{21}$$

Let $\overline{A_2}$ be the mean of an available time in an idle period. In order to find $\overline{A_2}$, we need to know what is the probability that a segment in an idle period is the maximum of two f-idle periods and what is the probability that a segment is simply a typical f-idle period. Since

$$b + c = \frac{1}{\tau_i + 2\tau_b}[\tau_i + 2\tau_b(1 - e^{-D/\tau_b})] < 1,$$

we need to normalize b and c. Let $b' = b/(b+c)$ and $c' = c/(b+c)$. Then

$$b' = \frac{\tau_i(1 - e^{-D/\tau_b})}{\tau_i + 2\tau_b(1 - e^{-D/\tau_b})}, \quad \text{and} \quad c' = \frac{\tau_i e^{-D/\tau_b} + 2\tau_b(1 - e^{-D/\tau_b})}{\tau_i + 2\tau_b(1 - e^{-D/\tau_b})}.$$

The mean of the maximum of two exponentially distributed lengths of f-idle periods is $3\tau_i / 2$[14]. Thus

$$\overline{A_2} = \tau_i + (\overline{Y} - 1)(\frac{3\tau_i}{2} b' + \tau_i c').$$

By replacing b', c', and $\overline{Y}$ in the equation above, we get

$$\overline{A_2} = \tau_i + \tau_i[(1 + \frac{\tau_i}{2\tau_b})e^{D/\tau_b} - 1] \times \frac{(1 - e^{-D/\tau_b})(3\tau_i/2 + 2\tau_b) + \tau_i e^{-D/\tau_b}}{\tau_i + 2\tau_b(1 - e^{-D/\tau_b})}. \tag{25}$$

Since $\overline{B_2} = \tau_b/2$, and $\overline{U_2} = D \cdot \overline{Y} = D(1 + \frac{\tau_i}{2\tau_b})e^{D/\tau_b}$,

$$SR_{COMP-D}(2) = 1 + \frac{\overline{B_2} + \overline{U_2}}{\overline{A_2}}. \tag{26}$$

V. Simulation Results

We built a simulator to evaluate the performance of the *MIG_D* protocol and the *COMP_D* protocol by using the language Maisie [15], and simulated both of protocols for a period of $1000\tau_i$. Our results are the means of 20 simulation runs. In all of our simulations, τ_i was set to 1.0, and τ_b and D are represented in multiples of τ_i. Fig. 6 and Fig. 7 show our analytic results and simulation results when D is $0.2\tau_i$ and τ_i respectively.

In both of figures, the curves labeled as 'Optimal Speedup' show $1 + \rho$, and the curves labeled as 'Analysis(M)' and 'Analysis(C)' represent our analytic results derived in the section IV. Our simulation results of the *MIG_D* protocol and the *COMP_D* protocol are represented by the curves labeled as 'Simulation(M)' and 'Simulation(C)' respectively.

Fig. 6 and Fig. 7 show that the delay for migration and state propagation can deteriorate the completion time of applications when the mean lengths of busy periods is small; f-idle periods

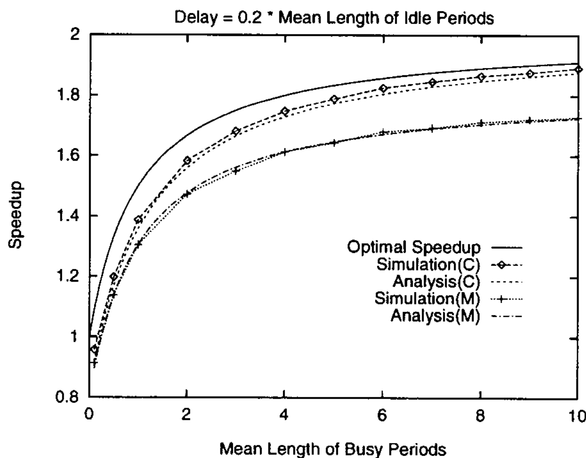


Fig. 6. Speedup with Two Processors when D=0.2τi.

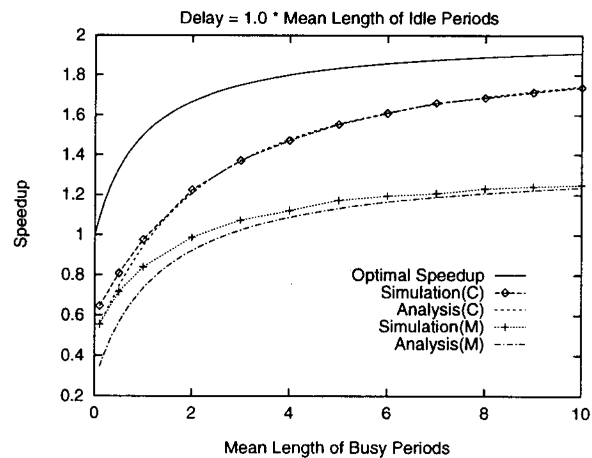


Fig. 7. Speedup with Two Processors when D = τi.

may be locally available before migration or state propagation is completed. In Fig. 7, the *COMP_D* protocol improves the performance when $\tau_b > \tau_i$ while the *MIG_D* protocol improves the performance when $\tau_b > 2.0\tau_i$. But we get an improved performance for the most of the region for both of protocols in Fig. 6.

When $\tau_b < D$, there is significant discrepancy between our simulation results and analytic results especially in Fig. 7. This occurs because we use $s = \tau_i / (\tau_i + 2\tau_b)$ in equation (9) approximately to derive analytic results in the *MIG_D* protocol and the *COMP_D* protocol respectively. Furthermore, the error of ω (= $1 + \tau_i / \tau_b$) in equation (17) is magnified for a given τ_i when τ_b approaches 0. However, for a given *D*, f-idle periods are likely to be locally available if τ_b approaches 0. Thus this is not the case we are interested in.

As is evident from Fig. 6 and Fig. 7, the performance difference between the *MIG_D* protocol and the *COMP_D* protocol increases as τ_b increases for a given *D*. The reason is that as τ_b increases in the *MIG_D* protocol, the average number of migration pushes in an idle period decreases and therefore the overhead of migration pulls becomes dominating. On the other hand, the *COMP_D* protocol does not suffer from any such overhead.

Similarly, as *D* increases, there is an increase in the overhead of migration pulls and a corresponding decrease in the probability of migration success. When a process migration fails, a process may have to migrate more than once without the chance of running in the *MIG_D* protocol. The probability of state propagation success also decreases in the *COMP_D* protocol, but each state propagation is guaranteed to improve the performance of applications because whenever the clone that is most ahead is preempted, competition protocols propagate its state to the other clones. Thus competition guarantees that the clone ahead of other clones is always on the fastest variable-speed processor among processors at any given time.

VI. Conclusions

We presented the reasons why competition protocols are preferable to migration protocols in a SNOW by using an example and qualitative arguments. Then we derived the closed form solutions for the speedup of the *MIG_D* protocol and the *COMP_D* protocol, and simulated both of the protocols to validate our analysis. By using analytic results and simulation results, we showed that competitive execution can finish sequential programs significantly faster than noncompetitive execution, especially when the foreground load on each processor is sufficiently high.

When the delay for migration and state propagation is non-zero, we demonstrated that competition protocols for sequential programs offer performance benefits that may be better than migration protocols under comparable overhead assumptions at the cost of redundant execution. Competition protocols use only otherwise idle CPU cycles, so redundant execution does not really cost at all. Of course, competitive execution cannot be employed when there is no enough idle CPU cycles. However, many researchers have observed that significant portion of CPU cycles on SNOWs has gone unused. In our companion paper[16], we will show that competition protocols offer even more speedup for distributed programs than for sequential programs.

References

[1] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team, "A Case for NOW(Networks of Workstations)", *IEEE Micro*, pp. 54-64, Feb. 1995.

[2] R. Agrawal and A. K. Ezza, "Location Independent Remote Execution in NEST", IEEE Trans. on Software Eng., Vol. 13, No. 8, pp. 905-912, Aug. 1987.

[3] R. E. Felderman, E. M. Schooler, and L. Kleinrock, "The Benevolent Bandit Laboratory: A Testbed for Distributed

Algorithms", *IEEE Journal on Selected Areas in Communications,* Vol. 7, No. 2, pp. 303-311, Feb. 1989.

[4] P. Krueger and R. Chawla, "The Stealth Distributed Scheduler", *11th Int'l Conf. on Distributed Computing Systems*, pp. 336-343, May 1991.

[5] M. J. Litzkow, M. Linvy, and M. Mutka, "Condor-A Hunter of Idle Workstations", *8th Int'l Conf. on Distributed Computing Systems*, pp. 104-111, 1988.

[6] D. A. Nicols, "Using Idle Workstations in a Shared Computing Environment", *Operating Systems Review*, Vol. 21, No. 5, pp. 5-12, Nov. 1987.

[7] G. C. Shoja, "A Distributed Facility for Load Sharing and Parallel Processing Among Workstations", *Journal of Systems and Software*", Vol. 14, pp. 163-172, 1991.

[8] M. Theimer, K. Lantz, and D. Cheriton, "Preemptable Remote Execution Facilities for the V System", *Proc. 10th ACM Symp. on Operating System Principles*, pp. 2-12, 1986.

[9] S. Zhou et al., "Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems", *Software: Practice and Experience*, Vol. 23, No. 12, pp. 1305-1336, Dec. 1993.

[10] S. H. Cho, *Competitive Execution in a Distributed Environment*, Ph.D. Dissertation, University of California, Los Angeles, Computer Science Department, 1995.

[11] L. Kleinrock, *Queueing Systems: Theory*, John Wiley and Sons, Inc., 1975.

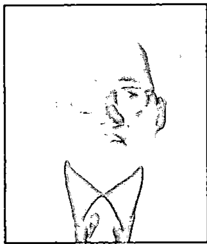
[12] D. R. Cox, *Renewal Theory,* Methuen and Co., Ltd., London, 1962.

[13] K. S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*, Prentice-Hall, Inc., 1982.

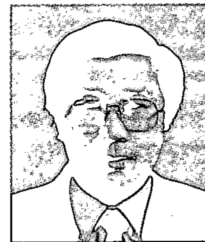
[14] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja, *A First Course in Order Statistics*, John Wiley and Sons, Inc., 1992.

[15] R. Bagrodia and W. T. Liao, "Maisie: A Language for the Design of Efficient Discrete-Event Simulations", *IEEE Software Eng.*, Vol. 20, No. 4, pp. 225-238, April 1994.

[16] S, H, Cho and S, S, Jun, "Competitive Execution of Distributed Programs on a Network of Shared Processors", accepted by *Journal of KISS(A): Computer Systems and Theory.*



Sung-Hyun Cho received the B.S. degree and the M.S. degree in computer science from Seoul National University in 1978 and 1980 respectively, and the Ph.D. degree in computer science from University of California at Los Angeles in 1995. He has served his military duty at Korea military academy from 1980 to 1983. He has worked as a postdoctoral fellow from 1995 to 1996. Since 1996, he has been with the division of computer and information communication in Hongik University. His research interests are in the areas on distributed operating systems, distributed database systems, distributed algorithms, realtime systems, and fault tolerance.



Sung-Syck Jun received the Ph.D. from Korea University in 1985. Since 1988, he has been a professor in the computer science department in Korea University. His research interests are in the areas on digital systems, computer architecture, and microprocessors.