Resource Allocation in Grid Computing

Ger Koole
Department of Mathematics
Vrije Universiteit
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands

Rhonda Righter
Department of Industrial Engineering and Operations Research
University of California
Berkeley, CA 94720
USA

Abstract

Grid computing, in which a network of computers is integrated to create a very fast virtual computer, is becoming ever more prevalent. Examples include the TeraGrid and Planet-lab.org, as well as applications on the existing Internet that take advantage of unused computing and storage capacity of idle desktop machines, such as Kazaa, SETI@home, Climateprediction.net, and Einstein@home. Grid computing permits a network of computers to act as a very fast virtual computer. With many alternative computers available, each with varying extra capacity, and each of which may connect or disconnect from the grid at any time, it may make sense to send the same task to more than one computer. The application can then use the output of whichever computer finishes the task first. Thus, the important issue of the dynamic assignment of tasks to individual computers is complicated in grid computing by the option of assigning multiple copies of the same task to different computers.

We show that under fairly mild and often reasonable conditions, maximizing task replication stochastically maximizes the number of task completions by any time. That is, it is better to do the same task on as many computers as possible, rather than assigning different tasks to individual computers. We show maximal task replication is optimal when tasks have identical size and processing times have a NWU (New Worse than Used; defined later)

0

distribution. Computers may be heterogeneous and their speeds may vary randomly, as is the case in grid computing environments. We also show that maximal task replication, along with a $c\mu$ rule, stochastically maximizes the successful task completion process when task processing times are exponential and depend on both the task and computer, and tasks have different probabilities of completing successfully.

1   Introduction

Grid computing, in which a network of computers is integrated to create a very fast virtual computer, is becoming ever more prevalent (Foster and Kesselman, 1999). Examples include networks such as the TeraGrid, a transcontinental supercomputer set up at universities and government laboratories and supported by NSF, applications on the existing Internet that take advantage of unused computing and storage capacity of idle desktop machines such as Kazaa, BitTorrent, SETI@home, stardust@home, Einstein@home, Climateprediction.net, and CERN's LHC@home (Large Hadron Collider), and ad hoc networks within universities or laboratories. Stanford's Folding@Home effort to understand the structure of proteins added Sony PlayStations to its grid in 2007, and it harnesses more computer power than all of the supercomputers in the world. Even Amazon is offering a grid computation service, EC2 (Elastic Computer Cloud). Computer makers are "grid-enabling" their new machines by implementing the Globus Toolkit (globus.org), a set of open-source software tools to support grid computing, and researchers are finding it easier to take advantage of "public computing" with new software platforms such as BOINC (Berkeley Open Infrastructure for Network Computing - boinc.berkeley.edu).

Grid computing creates a fast virtual computer from a network of computers by using their idle cycles. Although grid computing is a successor to distributed computing, the computing environments are fundamentally different. For distributed computing, resources are homogeneous and are reserved, leading to guaranteed processing capacity. On the other hand, grid environments are highly unpredictable. The computers are heterogeneous, their capacities are typically unknown and changing over time, and they may connect and disconnect from the grid at any time. For example, the statistical study of Dobber, van der Mei, and Koole (2006) found that for the Planetlab grid test bed environment, processing times of identical tasks on the computers of the grid showed a strong heterogeneity across different hosts and across time for the same host. In such an unpredictable environment, it may make sense to send the same task to more than one computer. The application can then use the output of whichever computer finishes the task first. As Cirne (2002) states for MyGrid, "The key is to avoid having the job waiting for a task that runs on a slow/loaded machine. Task replication is our answer for this problem." In addition to reducing total processing time on average, task replication is also very robust in terms of working well across a wide range of conditions, because unavailable or heavily loaded computers are basically ignored. Task replication is also easier and more practical to implement than alternative load-balancing schemes that attempt to monitor and predict the rapidly changing capacities of different computers. Yet another advantage of task replication is that tasks can be guaranteed to complete in FIFO (first-in first-out) order, making synchronization simple.

In our basic model with general processing time distributions, we assume that different tasks are identical in the sense that they will take the same amount of time to complete on some

canonical computer that is perfectly and totally available to process the task. For example, tasks may be performing the same subroutine on different data, as in Single-Program-Multiple-Data (SPMD) parallel programs. Thus, all randomness in task processing times comes from the computers because they may become more heavily loaded with higher priority (perhaps locally generated) jobs, and therefore the processing time for a task assigned to a computer is the same regardless of which task it is, and which tasks are assigned to other computers. SPMD programs are used in many applications, including computational fluid dynamics, environmental and economic modeling, image processing, and dynamic programming. Assuming that processing times depend on the computer and not the task is also reasonable for randomized algorithms, simulations, Monte-Carlo integration, and search engines.

There is an environmental state that may affect computer speeds and availabilities, as well as the arrival process. We assume that given the environmental state, the processing time of the same task on different computers is independent (as well as identically distributed). This is reasonable in a grid environment in which different computers are at different locations and belong to different entities. Processing times on the same computer may be dependent and can vary with the environmental state. Thus we can model the regime switching of computer speeds that was observed in planet lab data by Dobber, van der Mei, and Koole (2005).

We assume communication delays can be either be ignored, which is realistic when computation times are significantly longer than communication times, and is generally the case for applications in which grid computing makes sense, or they can be incorporated into the processing times.. We also assume that if a task is replicated and finishes on one computer, there is no cost for deleting that task from other computers. This assumption also seems reasonable in grid environments in which the reason one computer takes longer than another to process the same task is because it is either unavailable or busy with its own, higher priority, work. We only permit preemption or deletion of a task on a computer when that task completes on some other computer. The arrival process may be an arbitrary point process, as long as it is independent of the policy. We show that when task processing times have NWU (New Worse than Used) distributions, maximizing task replication stochastically maximizes the number of completed tasks by any time. That is, it is better to do the same task on as many computers as possible, rather than assigning different tasks to individual computers. NWU (and stochastic maximization) will be defined precisely later, but having NWU task processing times basically means that the time to finish a task that has just been assigned to a computer is stochastically smaller than the remaining time to finish a task that was assigned to a computer some time ago. This again is reasonable in a grid environment, where a computer that is taking a long time to process a task is probably unavailable or busy with other tasks, and will not get to our task for a while. We also show a complementary result, that when task processing times have NBU (New Better than Used) distributions, and there are two computers, we should not replicate tasks, except, perhaps, when there is only one task present. Note that our optimal

3

policies are independent of the environmental state, so they remain optimal even when the state is unobserved. Thus, monitoring costs may be reduced.

We also consider a model with exponential task processing times, and nonidentical tasks. (Exponential random variables are both NBU and NWU, and in particular, new and used are stochastically indistinguishable.) For this model the processing rates may vary across both computers and tasks, and may vary according to some environmental state. Also, tasks that complete may not complete correctly. All processing times are assumed to be independent (conditioned on the state), regardless of which task is assigned to which computer. That is, though some tasks may require more steps than others, and computers may vary in terms of their basic speeds, such that the mean time to process a task on a computer depends on both the task and the computer, the variability in processing times is again due to the computer and the environmental state, and not the task. For our exponential model, maximal replication of each task is again optimal. In addition, tasks should be ordered so that at any time the task with the largest product of success probability and intrinsic processing rate should be replicated on all computers (the $c\mu$ rule). In contrast, when processing times are geometric, Borst, Boxma, Groote, and Mauw (2003) showed that different tasks should be assigned to different computers as much as possible, i.e., task replication should be minimized, subject to using all available computers. We discuss this apparent contradiction later.

Another application of our model is in an R&D environment in which multiple teams may pursue the same research idea independently. In the presence of high variability across research teams, it may be optimal to have different teams simultaneously pursue the same (most promising) idea.

2 Heterogeneous Tasks with Exponential Processing Times

In the exponential case there is a set of parallel independent computers with different speeds, and tasks have different sizes, so that when a task of type i is processed on computer j its processing time is exponentially distributed with rate $\mu_i \nu_j(s)$, where s is an environmental state in some state space S. Roughly, we can think of $\nu_j(s)$ as the speed of computer j in state s and $1/\mu_i$ as the size of a task of type i. The environmental state includes information affecting the computer speeds and the arrival processes as described below, but is independent of which tasks are currently present and the policy. In some of these states some computers may be unavailable ($\nu_j(s) = 0$). The environmental state allows us to model the regime switching effect of computers that is observed in practice (Dobber, van der Mei, and Koole, 2006), as well as dependencies across computers. There is a chance that a task may not be completed correctly by a computer, for example, a randomized algorithm may not converge, or the machine is interrupted in a way that corrupts the data for the task. The probability of

successful completion for a task of type i in all states is c_i, and whether a given task is successful on a given computer is independent of any other successes of that task or other tasks. Arrivals form a general Markov arrival process (MAP). That is, there is an environmental continuous-time Markov chain with transition rates α_{xy} from state x to state y, and such that type i arrivals occur at x to y transitions of the Markov chain with probability β_{xy}^i. Such processes are dense in the class of arbitrary arrival processes (Asmussen and Koole, 1993). All other environmental state changes also occur according to Markov processes and are independent of the decisions made. We assume all transition rates are bounded with a common bound for all states. The same task may be assigned to more than one computer, in which case the task is considered complete the first time its processing time is finished on a computer and the completion on that computer is successful. If the completion is unsuccessful, the task is either lost (loss model), or can be restarted on the same computer (retrial model). When a computer becomes available, either because it has finished processing a task or because the environmental state has changed, any task, including copies of those already being processed on other computers, may be assigned to it. Processing times of the same task on different computers are assumed to be independent. Processing times on the same computer may be dependent; indeed, the state may include information on past processing times. Once the task is completed (loss model), or successfully completed (retrial model), all versions are immediately removed from the system. Let N_t be the cumulative number of successful task completions by time t.

We assume idling (not using an available computer) is permitted. We also assume that tasks may be preempted at any time without penalty. This requires a high-speed network, which is often the case, and is generally required for grid computing anyway. With these assumptions the optimal policy is the $c\mu$ rule, so the only time preemptions will occur is when tasks complete, or when a task with higher priority than any task present arrives, or when a computer becomes unavailable because of an environmental state change. For tasks of the same type, we may assume first-come first-served (FCFS) service, without loss of generality. Note that this policy is independent of the environmental state, so it is still optimal when the environmental state is unknown (and, in practice, the environment need not be monitored).

The optimality of the $c\mu$ rule for our model is consistent with existing results when task replication is not an option. For parallel-machine scheduling with exponential processing times and preemption permitted, the $c\mu$ rule (with the task with the highest $c\mu$ assigned to the fastest machine) is optimal for a variety of objective functions, interpretations of c (e.g., holding cost, reward for completion, probability of successful completion) and model extensions. See, e.g., Pinedo and Weiss (1980) and Liu and Righter (1997). The proof below can be modified to show that the $c\mu$ rule is optimal for our general model above, but without replication, and for our fairly general objective function. Because of the exponential processing times, for our model with parallel computers, always assigning the same task with the highest $c\mu$ on all available computers and taking the minimum processing time is essentially equivalent to creating a single

computer with speed equal to the combined speed of the available computers. Thus, given the optimality of maximal replication (so all computers are working on the same task), the optimality of the $c\mu$ rule for prioritizing tasks follows from existing results.

Let us recall some stochastic ordering definitions (e.g., Shaked and Shanthikumar, 1994). For two nonnegative continuous random variables X and Y, with respective distributions F and G, and $\bar{F}(x) = 1 - F(x)$ and $\bar{G}(x) = 1 - G(x)$, we say $X \geq_{st} Y$, i.e., X is stochastically larger than Y, if $E[h(X)] \geq E[h(Y)]$ for all increasing functions h, or equivalently, if $\bar{F}(t) \geq \bar{G}(t)$ for all $t \geq 0$. Also, $X \geq_{st} Y$ if and only if it is possible to construct two coupled random variables, $\hat{X}$ and $\hat{Y}$, so that $\hat{X} =_{st} X$ and $\hat{Y} =_{st} Y$ and $\hat{X} \geq \hat{Y}$ with probability 1. It is this last definition that we use in our proofs. For X_i an exponentially distributed random variable with rate λ_i, $X_i \geq_{st} X_j \iff \lambda_i \leq \lambda_j$, and for X_i a Bernoulli random variable with probability p_i, $X_i \geq_{st} X_j \iff p_i \geq p_j$, When we say a policy π stochastically maximizes the number of successful completions N_t at time t, we mean that for any other policy ρ, $N_t^\pi \geq_{st} N_t^\rho$, where N_t^π and N_t^ρ are the number of successful completions at time t under policies π and ρ respectively. Note that stochastically maximizing N_t $\forall t$ implies stochastically minimizing the makespan for a finite number of tasks. (We can set up the Markov arrival process so that arrivals stop after some given number of arrivals.) It also implies minimization of mean flow time.

We start with the loss model, in which unsuccessfully completed tasks are lost. In this case, over the evolution of the problem, we'll need to keep track of tasks completing successfully (for our objective function) and also those completing unsuccessfully (because they will no longer be available for processing).

Theorem 2.1 *For the loss model, in which unsuccessfully completed tasks are lost, the policy that never idles and always assigns the task with the largest $c_i \mu_i(s)$ to all available computers stochastically maximizes N_t for all $t \geq 0$.*

Proof. For simplicity we assume no environmental state, so the arrival rate of tasks is always λ, and $\mu_i(s) \equiv \mu_i$, $\nu_j(s) \equiv \nu_j$. The extension to a random environmental state, though notationally cumbersome, is straightforward.

We use uniformization, so we assume that (potential) events occur according to a Poisson process with rate $\lambda + \sum_j \nu_j \sum_i \mu_i$, and, without loss of generality, we set that rate equal to 1. Because we have a Markov system, we may assume without loss of generality that decisions are made only when (potential) events occur. Then, an event is an arrival with probability $\lambda = \lambda/(\lambda + \sum_j \nu_j \sum_i \mu_i)$, and, if a task of type i is being processed on computer j, the next event is the completion of task i with probability $\nu_j \mu_i$. With probability $1 - \lambda - \sum_j \nu_j \mu_{t(j)}$ the next event is a dummy event with no state change, where $t(j)$ is the task type currently

6

assigned to computer j, and where $\mu_{t(j)} = 0$ if no task is assigned to computer j. When a task of type i is being processed on computer j, we can think of the next event as being a *potential* task completion with probability $\nu_j \sum_i \mu_i$, and conditioned on there being a potential task completion, the probability that a task actually completes is $\mu_i / \sum_i \mu_i$. In the sequel we redefine μ_i to be $\mu_i / \sum_i \mu_i$ to ease notation. With uniformization we have essentially a discrete-time system, and we will call the times of potential events (arrivals or potential completions), i.e., the decision times, time 0, time 1, etc. The actual time of time k in the original system is the time of the k'th event in a Poisson process with rate 1. Let us condition on these actual event times and call the realized values $\sigma_k, k = 0, 1, \ldots$, with $0 = \sigma_0 < \sigma_1 < \sigma_2 < \cdots$.

Our proof is by induction on a finite time horizon T, where we assume the problem will stop at the time of the T'th event. Assume that the $c\mu$ rule is optimal for time horizon T (for $T = 0$ it is trivial), and consider horizon $T + 1$.

Suppose that at time 0 policy π puts some task 2 on some computer j when there is another task, task 1, with $c_1\mu_1 > c_2\mu_2$. We will show that following the $c\mu$ rule from time 0 to time $T + 1$ will be stochastically better (have stochastically more successful completions by time t for any t) than π. If π does not follow the $c\mu$ rule from time 1 to time $T + 1$, then we can construct a policy that agrees with π at time 0 (so they have the same task completions and states at time 1) and follows the $c\mu$ rule thereafter that will be stochastically better than π, from the induction hypothesis. Therefore, suppose π follows the $c\mu$ rule from time 1 on, so task 2 will not be processed again under π until task 1 completes. (Tasks with larger $c\mu$'s than task 1 may be processed before task 1 under π).

Let π' be an alternate policy (with N'_t the number of successful completions by time t) such that π' processes task 1 on computer j at time 0, and otherwise agrees with π at time 0. Let us couple the events, as well as their actual times, under policies π and π' so that $\sigma'_k = \sigma_k$ $\forall k$ and the same potential event (arrival or potential service completion) occurs under both π and π' for each k. If the coupled event at time 1 is not a potential completion for computer j, the states will be the same under both policies at that time, and letting π' agree with π from time 1 on, we have $N'_t = N_t$ for all t. Otherwise, if the coupled event at time 1 is a potential completion of computer j, let π' process task 2 whenever π is processing task 1, and let τ be the first time that some computer has a potential completion while π is processing task 1 (and π' is processing task 2). Refer to Figure 1 for a Gantt chart showing policies π and π'.

Define $I_1^i \sim Bernoulli(\mu_i)$ as an indicator for the event at time 1 being a completion (successful or not) of task i given that a potential completion on computer j occurred, $I_\tau^i \sim Bernoulli(\mu_i)$ as an indicator for the event at time τ being a completion of task i given that a potential task completion occurred (on any computer), and that i is being processed at time τ, and $J^i \sim Bernoulli(c_i)$ as an indicator for the completion of task i being successful given that

7

a potential completion occurs while i is being processed. Note that $I_k^i J^i \sim Bernoulli(c_i\mu_i)$ for $i = 1, 2$, $k = 1, \tau$.

For $t < \sigma_1$, $N_t' = N_t$. For $\sigma_1 \le t < \sigma_\tau$,

$$
\begin{aligned}
N_t &= S_t + I_1^2 J^2, \\
N_t' &= S_t + I_1^1 J^1,
\end{aligned}
$$

where S_t is the number of successful completions of tasks other than 1 or 2 (tasks with larger $c\mu$'s than $c_1\mu_1$) by time t given a j potential completion occurred at time 1. Thus, $N_t' \ge_{st} N_t$, for $0 \le t < \sigma_\tau$.

For $t \ge \sigma_\tau$,

$$
\begin{aligned}
N_t &= I_1^2 I_\tau^1 (J^1 + J^2 + A_t^{\{1,2\}}) + I_1^2 (1 - I_\tau^1)(J^2 + A_t^{\{2\}}) \\
&\quad + (1 - I_1^2) I_\tau^1 (J^1 + A_t^{\{1\}}) + (1 - I_1^2)(1 - I_\tau^1) A_t^{\emptyset} \\
&=_{st} I_t^2 I_1^1 (J^1 + J^2 + A_t^{\{1,2\}}) + I_\tau^2 (1 - I_1^1)(J^2 + A_t^{\{2\}}) \\
&\quad + (1 - I_\tau^2) I_1^1 (J^1 + A_t^{\{1\}}) + (1 - I_\tau^2)(1 - I_1^1) A_t^{\emptyset} \\
&= N_t'
\end{aligned}
$$

where $A_t^{\mathcal{S}}$ is the total number of successful completions of tasks other than 1 and 2 by time t given that tasks in $\mathcal{S}$ complete (either successfully or unsuccessfully) by time τ and those in $\{1, 2\} \backslash \mathcal{S}$ do not. That is, because of the way we have defined π', before time τ both policies will process the same (higher priority) tasks other than task 1, and given the information about whether tasks 1 and/or 2 are still in the system after time τ, the two policies will be the same from time τ on, so we can couple all the events so that $N_t' = N_t$ with probability 1, i.e., $N_t' =_{st} N_t$.

From the induction hypothesis, we can construct a policy that agrees with π' at time 0 and thereafter follows the $c\mu$ rule and that is stochastically better than π'. We can repeat the argument for all computers not assigned the task with the highest $c\mu$ at time 0, so we finally have that the $c\mu$ rule from time 0 to time $T + 1$ is stochastically better than any other policy. $\square$

It is not hard to modify the proof above to show that if all task completions are successful, but c_i is the reward earned upon completion of a task of type i, then the $c\mu$ rule maximizes ER_t for all t, where R_t is the total reward earned up to time t (essentially replacing J^i with its mean c_i).

In the retrial model, we need only keep track of successful completions of tasks, since unsuccessfully completed tasks remain in the system in the same state (with the same $c\mu$)

as before. Indeed, the model is equivalent to having all success probabilities equal to 1, but changing the parameters of the processing times from μ to $c\mu$. Thus, that the $c\mu$ rule stochastically maximizes N_t for all t follows from the theorem above. However, for the retrial model we can actually show a stronger result, that the $c\mu$ rule stochastically maximizes the *process* $\{N_t\} = \{N_t\}_{t=0}^{\infty}$, where N_t is the number of successful task completions by time t. When we say a policy π stochastically maximizes the process $\{N_t\}$, we mean that for any other policy ρ, $\{N_t^{\pi}\} \geq_{st} \{N_t^{\rho}\}$, that is, $P\{N_{t_1}^{\pi} \geq n_1, N_{t_2}^{\pi} \geq n_2, \ldots, N_{t_n}^{\pi} \geq n_k\} \geq P\{N_{t_1}^{\rho} \geq n_1, N_{t_2}^{\rho} \geq n_2, \ldots, N_{t_n}^{\rho} \geq n_k\}$ for any k and any $n_1, n_2, \ldots, n_k \geq 0$. Using an extension of the coupling definition above we will show that $\{N_t^{\pi}\} \geq_{st} \{N_t^{\rho}\}$ by constructing coupled processes $\{\hat{N}_t^{\pi}\} =_{st} \{N_t^{\pi}\}$ and $\{\hat{N}_t^{\rho}\} \geq_{st} \{N_t^{\rho}\}$ for any n and any $t_1, t_2, \ldots, t_n \geq 0$, with probability 1 $\hat{N}_{t_1}^{\pi} \geq \hat{N}_{t_1}^{\rho}, \hat{N}_{t_{12}}^{\pi} \geq \hat{N}_{t_2}^{\rho}, \ldots, \hat{N}_{t_n}^{\pi} \geq \hat{N}_{t_n}^{\rho}$. Indeed, our coupling will be such that *all* departures are earlier in one process than the other. This type of process stochastic maximization is also known as maximization across sample paths. Note that stochastic maximization of $\{N_t\}_{t=0}^{\infty}$ implies stochastic minimization of both the total flowtime up to any time t and the makespan for any finite number of tasks.

Theorem 2.2 *For the retrial model, in which unsuccessfully completed tasks are lost, the policy that never idles and always assigns the task with the largest $c_i\mu_i(s)$ to all available computers stochastically maximizes $\{N_t\}_{t=0}^{\infty}$.*

Proof. The proof is along the same lines as above, so we focus on the differences. Here the $c\mu$ rule corresponds to having all computers process the task with the highest $c\mu$ until the task successfully completes (or it is preempted by a higher priority task). Again we use uniformization and induction on the time horizon and we suppose that at time 0 policy π puts some task 2 on some computer j when there is another task, task 1, with $c_1\mu_1 > c_2\mu_2$. We also define π' as before, so that we only have a difference between the two policies if the event at time 1 is a potential completion of computer j. With τ as defined before, we can show that the any successful completions at times 1 and τ are *jointly* earlier, using the following coupling. (All events at times other than 1 and τ are the same for both policies.) Let $I_1^i \sim Bernoulli(c_i\mu_i)$ be an indicator for the event at time 1 being a *successful* completion of task i given that a potential completion on computer j occurred and that i is being processed on computer j at time 1, and let $I_{\tau}^i \sim Bernoulli(c_i\mu_i)$ be an indicator for the event at time τ being a *successful* completion of task i given that a potential task completion occurred, and that i is being processed at time τ. Then

$$N_t = I_1^2 I_{\tau}^1 (2 + A_t^{\{1,2\}}) + I_1^2(1 - I_{\tau}^1)(1 + A_t^{\{2\}}) + (1 - I_1^2)I_{\tau}^1(1 + A_t^{\{1\}}) + (1 - I_1^2)(1 - I_{\tau}^1)A_t^{\emptyset}$$

$$N_t' = I_t^2 I_1^1 (2 + A_t^{\{1,2\}}) + I_{\tau}^2(1 - I_1^1)(1 + A_t^{\{2\}}) + (1 - I_{\tau}^2)I_1^1(1 + A_t^{\{1\}}) + (1 - I_{\tau}^2)(1 - I_1^1)A_t^{\emptyset}$$

9

where $A_t^{\mathcal{S}}$ is the total number of successful completions of tasks other than 1 and 2 by time t given that tasks in $\mathcal{S}$ *successfully* complete by time τ and those in $\{1,2\}\backslash\mathcal{S}$ do not. Now, we couple the indicators under the two policies as follows. Let

$$
\begin{array}{lll}
\hat{I}_1^2 = \hat{I}_1^1 = \hat{I}_\tau^1 = \hat{I}_\tau^2 = 1 & \text{with probability} & c_1\mu_1 c_2\mu_2, \\
\hat{I}_1^2 = \hat{I}_1^1 = \hat{I}_\tau^1 = \hat{I}_\tau^2 = 0 & \text{with probability} & (1-c_1\mu_1)(1-c_2\mu_2), \\
\hat{I}_1^2 = \hat{I}_1^1 = 1, \hat{I}_\tau^1 = \hat{I}_\tau^2 = 0 & \text{with probability} & c_2\mu_2(1-c_1\mu_1), \\
\hat{I}_1^2 = \hat{I}_1^1 = 0, \hat{I}_\tau^1 = \hat{I}_\tau^2 = 1 & \text{with probability} & c_2\mu_2(1-c_1\mu_1), \\
\hat{I}_1^2 = \hat{I}_\tau^2 = 0, \hat{I}_1^1 = \hat{I}_\tau^1 = 1 & \text{with probability} & c_1\mu_1 - c_2\mu_2.
\end{array}
$$

With this coupling, either there are successful completions at both times 1 and τ under both policies, or there are no successful completions at either times 1 or τ under both policies, or there is exactly one successful completion at either times 1 or τ under both policies. In the latter case, either the successful completion occurs at time 1 for both policies, or at time τ for both, or it occurs at time 1 under π' and time τ under π. Note that our coupling is legitimate, i.e., the probabilities are correct on the margin for each policy, because

$$
\begin{array}{llll}
P\{\hat{I}_1^2 = \hat{I}_\tau^1 = 1\} & = P\{\hat{I}_\tau^1 = \hat{I}_1^2 = 1\} & c_1\mu_1 c_2\mu_2 \\
P\{\hat{I}_1^2 = \hat{I}_\tau^1 = 0\} & = P\{\hat{I}_\tau^1 = \hat{I}_1^2 = 0\} & (1-c_1\mu_1)(1-c_2\mu_2) \\
P\{\hat{I}_1^2 = 0, \hat{I}_\tau^1 = 1\} & = P\{\hat{I}_\tau^1 = 1, \hat{I}_1^2 = 0\} & c_1\mu_1(1-c_2\mu_2) \\
P\{\hat{I}_1^2 = 1, \hat{I}_\tau^1 = 0\} & = P\{\hat{I}_\tau^1 = 0, \hat{I}_1^2 = 1\} & (1-c_1\mu_1)c_2\mu_2
\end{array}
$$

Since all other completions occur at the same times under both policies, we have $\{N_t^{\pi'}\}_{t=0}^\infty \geq \{N_t^\pi\}_{t=0}^\infty$ with probability 1 (across the whole sample path). The rest of the argument is as before. $\square$

It is easy to see that if we have an extra resequencing constraint, that is, that the outputs of tasks must be used in the same order as the tasks are ordered (e.g., FIFO), and if tasks are identical, replicating tasks as much as possible will still be optimal, because this guarantees that tasks complete in order. The same holds true of programs that consist of sequential sets of parallelizable tasks, where synchronization must occur for each set of tasks before the next set can start.

It is also not hard to show that if preemption and idling are not permitted, and the c_i's are the same for all tasks, the "μ rule" (or SEPT, shortest expected processing time first), of assigning the stochastically shortest task to all computers, is optimal.

At first surprisingly, our result for exponential processing times is the opposite of the result for the geometric case (Borst et al., 2003). For identically and geometrically distributed processing times with success probabilities equal to 1, Borst et al. have shown that the optimal policy assigns different tasks to different computers whenever possible, and when there are fewer tasks than computers, though all computers should be used, each task should have the

minimum number of copies possible. Of course in the exponential case when tasks are identical in both c and μ, all assignment rules that use all available computers are stochastically identical (so minimal task replication is also optimal in the exponential case). Also, in a discrete model such as that of Borst et al., it is possible for several computers to finish at the same time, and it is wasteful to have them finish the same task, so there is an incentive to minimize replications.

3 Identical Tasks with Generally Distributed Processing Times

Now we suppose the tasks are identical, so the only question is whether to process multiple copies of the same task on different computers. We assume that nominal task processing times and probabilities of successful completion are independent of the state and policy, though other processes may depend on an environmental state. The processing time of a task on a computer is independent of which task it is and which tasks are assigned to other computers. We first suppose that the common probability of successful completion given completion of a task on a computer is 1. Arrivals of tasks may follow an arbitrary stochastic process, as long as it is independent of the policy, and computers may have different, finite, speeds that can vary according to arbitrary stochastic processes, again independent of the policy. When a task completes on a computer, all copies of the task are immediately removed from the system. Otherwise, tasks may not be preempted once assigned to a computer. Idling is permitted.

Note that in the presence of processing time variability, task replication is appealing because, if all computers are processing the same task, as soon as the first one finishes, all computers become available to process more tasks. Our results are consistent with this intuition.

We first define and develop intuition for the concepts of new better or worse than used. See Shaked and Shanthikumar [8] or Müller and Stoyan [7] for details and further background.

3.1 NWU (NBU) Peliminaries

Let X be a random task processing time on a computer whose speed is always 1. We call X the nominal processing time, and assume that its distribution is continuous and identical for all tasks. Let $X_t = \{X - t | X > t\}$ be the remaining processing time of a task that has completed t time units of processing, and let $\bar{F}(x) = P\{X > x\}$. We say that X is New Worse than Used (NWU) if the remaining processing time of a task that has received some processing (is used) is stochastically larger than the processing time of a task that has received no processing (is new), i.e., $X_0 \leq_{st} X_t$ for all t, or equivalently, $\bar{F}(x+y) \geq \bar{F}(x)\bar{F}(y)$ for all x, y. Note that the "worse" comes from reliability theory, in which it is worse to have component lifetimes that are short. In a scheduling context, it is just the opposite, i.e., short task processing times are better, but we stick with well-established terminology. An equivalent definition for NWU is to say that

11

for any t we can construct coupled versions, $\hat{X}_0 =_{st} X_0$ and $\hat{X}_t =_{st} X_t$, so that $\hat{X}_0 \leq \hat{X}_t$ with probability 1. Note that under our assumptions on the computer speed processes, $X_0 \leq_{st} X_t$ implies that $C_j(X_0, u, S(u)) \leq_{st} C_j(X_t, u, S(u))$ for any time u, where $C_j(Y, u, S(u))$ is the actual completion time of a task started at time u on computer j when the state of the system is $S(u)$. NBU (New Better than Used) distributions are defined analogously, with analogous properties.

A sufficient condition for X to be NWU is for X to have decreasing failure rate (DHR), because this is equivalent to X_t stochastically increasing in t. An example of a DHR distribution is the hyperexponential distribution. If a processing time is DHR then, roughly, the longer the task has been worked on, the less likely it is to finish soon. In our context this may be a very reasonable assumption, because a computer may either process the task quickly or take a long time, depending on its workload of other tasks for other users. Similarly, if X is IHR (has increasing hazard rate), then X is NBU. An example of an IHR distribution is the Erlang distribution.

Intuitively, NWU distributions are more variable than NBU distributions. For example, the coefficient of variation of an NWU random variable is at least 1, while it is at most 1 for an NBU random variable. Of course exponential random variables, with a coefficient of variation of 1, are both NBU and NWU.

To make the ideas of NWU, i.e., $X_0 \leq_{st} X_t$, and coupling concrete, consider the following example of a mixture of two exponentials (a hyperexponential): $X = X_0 = IY_s + (1 - I)Y_b$, where $I \sim Bernoulli(1/2)$, $Y_s \sim \exp(3)$, $Y_b \sim \exp(1)$ (s for small, b for big). At time 0, X_0 is equally likely to be the small or the big exponentially distributed random variable. Now suppose that the task with initial processing time $X = X_0$ has completed 2 time units of processing and still has not completed. Then $X_2 = I'Y_s + (1 - I')Y_b$, where $I' \sim Bernoulli(p)$, and where

$$
\begin{aligned}
p &= P\{X = Y_s | X > 2\} = P\{X = Y_s, X > 2\}/P\{X > 2\} \\
&= \frac{1}{2}e^{-(3)(2)} / \left(\frac{1}{2}e^{-(3)(2)} + \frac{1}{2}e^{-(1)(2)} \right) \approx .02.
\end{aligned}
$$

Thus, after completing 2 units of processing, the remaining processing time has only a 2% chance of being the small random variable. We can couple the random variables so that $\hat{X}_0 \leq \hat{X}_2$ with probability 1 as follows. With probability .02 let $\hat{I} = \hat{I}' = 1$; with probability .50 let $\hat{I} = \hat{I}' = 0$; with probability .48 let $\hat{I} = 0$ and $\hat{I}' = 1$, so $\hat{I} \sim Bernoulli(.50)$ and $\hat{I}' \sim Bernoulli(.02)$. Let $\hat{Y}_s \sim \exp(10)$ and let $\hat{Y}_b = 10\hat{Y}_s$, so

$$
P\{\hat{Y}_b > t\} = P\{10\hat{Y}_s > t\} = P\{\hat{Y}_s > t/10\} = P\{Y_s > t/10\} = e^{-10t/10} = P\{Y_b > t\}.
$$

Then $\hat{X}_0 = \hat{I}\hat{Y}_s + (1 - \hat{I})\hat{Y}_b \leq \hat{I}'\hat{Y}_s + (1 - \hat{I}')\hat{Y}_b = \hat{X}_t$ with probability 1. Note that it doesn't matter that $\hat{Y}_s$ and $\hat{Y}_b$ are dependent because $\hat{X}_0$ and $\hat{X}_t$ only use one or the other of $\hat{Y}_s$ and $\hat{Y}_b$.

3.2   Results for NWU Processing Times

Suppose processing times are NWU. Then the optimal policy maximizes replications, i.e., it is optimal to always assign the same task to all computers and to never idle. Let us call this policy the MRNI (maximal replications, non-idling) policy.

We say a task is a "fresh" task if it has not yet been assigned to any computer, and it is an "old" task if some copies of it have already been assigned. Note that any time a task is assigned to a computer, regardless of whether it is fresh or old or how many copies of the task are currently running, the processing time from the point of assignment is X_0. This is intuitively why, for NWU processing times, we prefer to assign old tasks; because their remaining processing times on other computers are getting longer, and when we replicate the old task we have a chance of a short (new) processing time that will eliminate all outstanding copies of the task, freeing up multiple computers. The lemma below makes this intuition rigorous, where N_t is the total number of task completions by time t.

Lemma 3.1 *If processing times are NWU, then it is never optimal to assign a fresh task when old tasks are present. More specifically, for any policy that assigns a fresh task when old tasks are present, we can construct a policy that assigns old tasks, such that $\{N_t\}_{t=0}^{\infty}$ is stochastically larger under the new policy.*

Proof. Let π be an arbitrary policy that at some time, call it time 0, assigns a fresh task, call it task 2, to a set of computers when an old task, call it task 1, is present. Let π' agree with π starting at time 0 except that whenever π assigns task 2 to a computer (call such computers A-computers) π' assigns task 1, until one of the computers with task 1 assigned to it under π' completes, at time τ say. The corresponding computer under π could be processing either task 1 (case 1) or task 2 (case 2, if the computer is an A-computer). Refer to Figure 2 for a Gantt chart illustrating an 8-computer example, where the 4th, 5th and 6th computers are A-computers, and where a bold line on the right of a processing time block means that the corresponding computer is the one that completed the corresponding task first. (Other computers with the same task stop processing the task at the same time.) Let the remaining processing times of all tasks currently being processed at time 0 be the same for both policies, and whenever a task is assigned to a computer under either policy between times 0 and τ, let the processing time of the task on that computer be the same for both policies (regardless of

13

whether the policies assign the same task; recall that task processing times are stochastically identical). Between times 0 and τ, task 2 is not processed under π' by construction, and neither task 1 nor task 2 completes under either policy, by definition of τ. If the completing computer at time τ also has task 1 assigned to it under π (Case 1), then there is a task 1 departure for both policies, all computers except the A-computers will be in the same state for both policies, and the A-computers will be available under π' but not under π. Let π' assign task 2 to the A-computers, and suppose their (new) processing times are coupled with the remaining (used) processing times on these computers under π so that they are smaller under π' (which we can do because processing times are NWU). These processing times are shown with dotted lines in Figure 2. Let π' otherwise agree with π from time τ until either task 2 completes on some computer other than an A-computer (Case 1a), in which case the two policies will be in the same state, or an A-computer completes under π' (Case 1b). In the latter case π' has a task completion, but π does not. Let π' agree with π except that it idles computers on which task 2 is assigned under π, until task 2 completes under π, at time σ say. At this point both policies are in the same state, and all departures are the same under both policies, except that task 2 departs earlier under π'.

Note that π' must be able to observe the state under π when an A-computer completes after time τ, so that it can idle until task 2 completes under π. However, the remaining time until task 2 completes is independent of all the other random variables in the system operating under π, so π' is still non-anticipative.

If the completing computer at time τ is an A-computer (case 2), then both policies have a departure (task 1 under π' and task 2 under π). Let us relabel the remaining task under π so that it is called task 2 under both policies. Recall that all tasks are identical, so the relabeling is valid. Let us also now call the computers that have task 2 assigned to them under π (and are available under π') the A-computers. The rest of the argument is then the same as in case 1. □

Theorem 3.2 *If processing times are NWU and we start with no old tasks, then the MRNI policy stochastically maximizes $\{N_t\}_{t=0}^{\infty}$.*

Proof. From the lemma above we need only show that when there are no old tasks initially, and when fresh tasks are only started when all old tasks are complete, it is never optimal to idle. Let π be a policy that sometime idles but otherwise never starts fresh tasks when an old task is present. That is, π always has the same task on all computers until it completes. Let π' never idle and always assign the same task to all computers. Let T_i (T_i') be the time of the ith departure, or task completion, under π (π'), with $T_0 = T_0' = 0$, and let $Y_{i,j} =_{st} X$ be the

nominal processing time of task i on computer j. Then

$$
\begin{aligned}
T'_i &= \min_j\{C_j(Y_{i,j}, T'_{i-1}, S(T'_{i-1}))\} =: \min_j V'_{ij}, \\
T_i &= \min_j\{C_j(Y_{i,j}, T_{i-1} + \delta_{ij}, S(T_{i-1} + \delta_{ij}))\} =: \min_j V_{ij},
\end{aligned}
$$

where V_{ij} would be the completion time of task i on computer j if it were the only computer from time T_{i-1} on, and δ_{ij} is the amount of time computer j idles before starting the i'th task under π, which could be a random variable. (Recall that $C_j(Y, u, S(u))$ is the actual completion time of a task with nominal processing time Y started at time u on computer j when the state of the system is $S(u)$.) Thus, we will have stochastically earlier departures under π' by induction on i if we can show, for any j and $i \geq 1$, that $V_{ij} \leq_{st} V'_{ij}$ whenever $T'_{i-1} \leq_{st} T_{i-1}$. Let us fix i and j, and couple $T'_{i-1} \leq_{st} T_{i-1}$ and condition on their values so that $T'_{i-1} = t' \leq T_{i-1} = t$. Let us also condition on the state at time t' and the processes controlling the speed of all the computers from t' on, so that the completion time of a task started at time t' on j is an increasing deterministic function, f, of its processing time, σ. Let us condition on $\delta_{ij} = d$, and let σ_0 be such that $f(\sigma_0) = d + t$, i.e., it is the processing time of a task that if started on computer j at time t' would complete at time $t + d$. We also condition on $Y_{i,j} = y$, for both policies. If $y \leq \sigma$, then $V'_{i,j} = f(y) \leq f(\sigma_0) = t + d \leq V_{i,j}$. Otherwise, $V'_{i,j} = f(y) \leq f(\sigma_0 + y) = V_{i,j}$. Therefore, $T'_i = \min_j V'_{ij} \leq \min_j V_{ij} = T_i$. $\square$

Note that part of our proof does not depend on the distribution of processing times. In particular, we showed that when the same task is always assigned to all computers then there should be no unnecessary idling, for any task processing time distribution, and the distribution may depend on the task.

Now suppose that tasks have the same probability of successful completion c, but $c < 1$, and we let $\hat{N}_t$ be the number of successful task completions by time t; N_t is still the number of task completions, whether successful or not. As in the last section, we consider both a loss model, in which unsuccessfully completed tasks are lost, and a retrial model, in which tasks can be started again after unsuccessful completions until they are successfully completed.

Corollary 3.3 *If processing times are NWU and tasks have a common probability of success $c < 1$, and we start with no old tasks, then the MRNI policy stochastically maximizes $\{\hat{N}_t\}_{t=0}^{\infty}$, for both the loss and retrial models.*

Proof. The argument in the proof of Theorem 3.2 also shows that MRNI stochastically maximizes $\{N_t\}_{t=0}^{\infty}$ even when $c < 1$, for both the loss and retrial models. Also, for both

models $\hat{N}_t = \sum_{k=1}^{N_t} I_{(k)}$, where $I_{(k)} \sim Bernoulli(c)$ (i.i.d.) is the indicator for successful completion of the kth task to complete. Since success probabilities are the same for all tasks, we have no preference for ordering tasks, and $\{\hat{N}_t\}_{t=0}^{\infty}$ is stochastically maximized if $\{N_t\}_{t=0}^{\infty}$ is stochastically maximized. $\square$

Now suppose that tasks have different success probabilities, c_i, but they still have stochastically identical processing times. Suppose there are K fresh tasks and no old tasks initially. With nonidentical tasks we must now also assume there are no arrivals.

Corollary 3.4 *If processing times are NWU, and there is a fixed set of fresh tasks with different c_i's, and no arrivals and no old tasks, then the policy that always assigns the uncompleted task with the largest c_i to all computers and never idles stochastically maximizes $\{\hat{N}_t\}_{t=0}^{\infty}$, for both the loss and retrial models.*

Proof. As observed in the last proof, we know that MRNI stochastically maximizes $\{N_t\}_{t=0}^{\infty}$, and $\hat{N}_t = \sum_{k=1}^{N_t} I_{(k)}$, where $I_{(k)} \sim Bernoulli(c_{(k)})$ (i.i.d.) is the indicator for successful completion of the kth task to complete. Also, because there are no arrivals, to stochastically maximize $\{\hat{N}_t\}_{t=0}^{\infty}$ we will want to stochastically maximize $\{N_t\}_{t=0}^{\infty}$ (so follow the MRNI policy), so we need only determine the order in which to do the K fresh tasks. (If we had arrivals, then if at some time we had only tasks with small c's present, we might want to idle to wait for a task with a higher c.) A coupling and interchange argument along the lines of the proof of Theorem 2.2 shows that the optimal order is largest c first. $\square$

If the tasks have different rewards, c_i, rather than success probabilities, the argument above for the loss model shows that the c rule stochastically maximizes the cumulative reward process when we start with a fixed set of fresh tasks and there are no arrivals.

3.3 Results for NBU Processing Times

Now we assume that X is NBU, so that the remaining processing time of a task that has just been started is larger than one that has been worked on for a while, and its coefficient of variation is at most 1. We also assume that there are only two stochastically identical computers, tasks are stochastically identical, and, for simplicity, that there is no environmental state. In this case task replication should be minimized when there are at least two tasks in the system, and we should never idle. This means that if there is only one task it should be processed on both computers, but it might be preempted by an arriving task on one of them. Note that for this model with arrivals, because we don't allow preemption except at task completions, if there are no fresh tasks and if one computer is working and one is idle, it may be optimal to keep the idle computer idle until there is an arrival, rather than replicating

the old task on the idle computer. This will be true for example when processing times are deterministic.

Theorem 3.5 *If processing times are NBU and there are only two computers, to stochastically maximize $\{N_t\}_{t=0}^{\infty}$, fresh (different) tasks should be assigned to an available computer whenever possible, there should be no idling when at least two tasks are present, and the computers should never both be idle when any task is present.*

Proof. We first show that we should never idle both computers. Suppose some policy π does idle both computers when some task, task 1 say, is present. Let δ be the time π first assigns a task, say task 1 without loss of generality, to a computer; call it computer 1. Let π' process task 1 on computer 1 at time 0, and let it otherwise agree with π until task 1 completes under π'. Suppose the processing time of task 1 on computer 1 under both policies is X, and condition on $X = x$. Let all other processing times be the same for both policies. Refer to Figure 3. If task 1 is also processed on computer 2 and completes before x for both policies (case 1), then the states will be the same and letting π' agree with π thereafter, π and π' will have the same task completion processes. Otherwise, task 1 will complete earlier under π' (at time x) than under π, and any task completions (of other tasks) before time x on computer 2 will be the same for both policies (case 2). Let $\gamma \leq \delta + x$ (and $\gamma > x$) be the completion time of task 1 under π, and let π' idle computer 1 from time x to time γ and, if $\gamma < \delta + x$ (so task 1 completes on computer 2 at time γ under π) let π' also idle computer 2 from time x to time γ, and otherwise let π' agree with π, so all completions besides that of task 1 will be the same for both policies. That is, $\{N_t^{\pi'}\}_{t=0}^{\infty} \geq \{N_t^{\pi}\}_{t=0}^{\infty}$ with probability 1. We can repeat this argument to show that never idling both computers before time T is stochastically better than idling them before T, for any arbitrarily large T.

Now suppose π assigns an old task, say task 2, to a computer, say computer 1, when a fresh task is available (so computer 2 is processing task 2). Let π' assign a fresh task, call it task 1, to computer 1, and agree with π for computer 2. Let $X_1 =_{st} X$ be the processing time of the task assigned to computer 1 under both policies (we can do this because processing times have the same distribution for all tasks), let $R_2 =_{st} X_t$ be the remaining processing time of the task on computer 2 under both policies where t is the amount of processing that task 2 has already received on computer 2, and let $\gamma = \min(X_1, R_2)$ be the time of the first task completion for both policies. Then under π task 2 completes at time γ and both computers are available, and under π' one of tasks 1 and 2 completes while the other computer may still be processing a task. Let us (possibly) relabel the tasks and computers under π' so that we call task 2 the task that completes at γ and task 1 the one that (may) still be being processed on computer 1. Since both computers are idle at time γ under π, from the argument above we may assume that π will assign a task, call it task 1 without loss of generality, to a computer, call it computer 1,

17

at time γ. (It may also assign a task to computer 2.) Let π' agree with π for any assignments to computer 2.

Let $\hat{X} =_{st} X$ be the processing time of task 1 on computer 1 under policy π, and let $\hat{R} =_{st} X_\gamma$ be the remaining processing time of task 1 on computer 1 under policy π' and let them be coupled so that $\hat{R} \leq \hat{X}$ with probability 1 (because X is NBU). If π (and π') assign task 1 to computer 2 before $\gamma + R$, then, because processing times on different computers are independent, task 1 could complete on computer 2 before $\gamma + R$. In this case, at the time task 1 completes the states under both policies will be the same, and letting π' agree with π from then on, $\{N_t^{\pi'}\}_{t=0}^\infty =_{st} \{N_t^\pi\}_{t=0}^\infty$. Suppose task 1 does not complete on computer 2 before $\gamma + R$. Then at time $\gamma + R$ task 1 completes under π' but is still running under π. Let π' idle computer 1 from time $\gamma + R$ until task 1 completes under π, and let it otherwise agree with π. Then when task 1 completes under π the states under the two policies will be the same, so they will have the same task completion times except for task 1, which completes earlier under π'. We will show next that if there are fresh tasks available at time $\gamma + R$, then there is a policy that does not idle at time γ_2 that is better than π' and hence better than π.

Finally, suppose π idles computer 2 when there is a fresh task, call it task 2, that is available, and suppose, because we've already shown it is not optimal to idle both computers, that task 1 is being processed on computer 1. Let π' assign task 2 to computer 2. We've shown that it is optimal to always assign fresh tasks when possible, so we can assume without loss of generality that π will assign task 2 to computer 2 when it finishes idling, at time δ say. Let x be the processing time of task 2 on computer 2 under both policies. Let $\gamma \leq \delta + x$ be the time task 2 completes under π. (We could have $\gamma < \delta + x$ if task 2 is processed on computer 1). Let π' agree with π for assignments to computer 1 up to time γ, and let it idle computer 2 from time x to time γ if $x < \gamma$. At time γ the states will be the same under the two policies, so letting π' agree with π from then on, all task completions will be the same under the two policies, except that task 2 may complete earlier under π'. Again we can repeat the argument to show that never idling when a fresh task is available is better than a policy that does so idle. $\square$

4 Conclusions

We have found that when processing times have high variability, in the sense that tasks that have been worked on for a while have longer remaining processing times than tasks that have received no processing (and therefore have coefficients of variation of 1 or larger), then it is optimal to process the same task on as many computers as possible. This task replication allows us to take advantage of the chance of small processing times, and it means that more computers will be available when a task completes. It also has the advantage of creating a FIFO (first-in first-out) ordering of tasks, which is helpful in synchronizing large, complicated programs. Another

advantage is that it is independent of the state of the system, and therefore expensive monitoring and load balancing procedures may be avoided. Interesting open questions are conditions under which task replication is a good idea even when there are penalties for stopping unfinished tasks, or when all copies of a task must be processed to completion.

Another research direction is to investigate good policies when processing times are variable (e.g., they still have large coefficients of variation), but are not necessarily NWU. It is known that a Gittins' index policy is optimal for a single computer and general processing times with preemption, and that such a policy is approximately optimal with parallel processors in heavy traffic. Future research may provide good heuristics for replicating tasks using Gittins' indices. We are also investigating good policies based on specific processing time distributions. For example, data indicates that a mixture of normal distributions may be a reasonable approximation for processing times. In these cases, we expect policies that are intermediate between maximal and minimal replication to be good; e.g., when there are 100 computers, replicate each task 5 times, as long as there are at least 20 tasks.

5   Acknowledgements

References

[1] S. Asmussen and G. Koole, Marked point processes as limits of Markovian arrival streams, J. Appl. Prob. vol. 30, pp. 365-372, 1993.

[2] S. Borst, O. Boxma, J.F. Groote, S. Mauw, Task allocation in a multi-server system. J. of Scheduling. vol. 6, pp. 423-436, 2003.

[3] W. Cirne, MyGrid: A user-centric approach for grid computing. Walfredo.dsc.ufcg.edu.br/talks/MyGrid.ppt, 2002.

[4] M. Dobber, R. van der Mei, and G. Koole, Statistical properties of task running times in a global-scale grid environment. Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), pp. 150-153, 2006.

[5] L. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann: San Francisco, CA, 1999.

[6] Z. Liu and R. Righter, Optimal Scheduling on Parallel Processors with Precedence Constraints and General Costs, *Prob. in the Eng. and Inform. Sci.* , vol. 11, pp. 79-93, 1997.

[7] A. Müller and D. Stoyan, *Comparison Methods for Stochastic Models and Risks*. New York, Wiley, 2002.

[8] M. Shaked and J.G. Shanthikumar, *Stochastic Orders*. New York: Academic Press, 1994.

[9] G. Weiss and M. Pinedo, Scheduling Tasks with Exponential Service Times on Non-Identical Processors to Minimize Various Cost Functions, J. Appl. Prob. vol. 17, pp. 187-202, 1980.