Parallel application performance in a shared resource environment

Distrib. Syst. Engng **3** (1996) 9–19. Printed in the UK

Parallel application performance in a shared resource environment

Gregory D Peterson† and Roger D Chamberlain‡

Computer and Communications Research Center, Washington University, Campus Box 1115, One Brookings Drive, St. Louis, MO, 63130-4899, USA

Abstract. The utilization of networked, shared, heterogeneous workstations as an inexpensive parallel computational platform is an appealing idea. However, most performance models for parallel computation are oriented towards the use of tightly-coupled, dedicated, homogeneous processors. We develop and validate an analytic performance modelling methodology for synchronous iterative algorithms executing on networked workstations. The model includes the effects of application load, background load, and processor heterogeneity. We use two applications, nonlinear optimization and discrete-event simulation, to validate the model. Various policies for the use of the workstations are considered and the optimal (or near-optimal) scheduling found. The performance modelling methodology provides significant help in addressing scheduling and similar issues in a shared resource environment.

1. Introduction

To provide cost-effective computing resources for computationally intensive applications, parallel processing techniques are increasingly being applied to networks of existing workstations. The majority of the time, these workstations are idle and therefore under-utilized. The use of networked workstations as a parallel computing platform raises a number of interesting issues, especially when the primary use that motivated a workstation's purchase was the day-to-day computing needs of the workstation's owner.

The processing environment assumed here is a network of workstations that are connected via a local area network. The workstations are not dedicated resources; several users may be utilizing them while the computation of interest is executing. In addition, the power (i.e., computational speed) of the individual workstations may vary, although we will assume their basic architecture is the same (single CPU, significant local memory, possibly local disk). In order to facilitate cooperative work across the workstations, there are a number of systems available that provide message passing and process control primitives [26]. Our experimental results use the Parallel Virtual Machine (PVM) system [24].

The use of networked workstations as a distributed computing platform has many similarities with massively parallel processing (MPP) systems (e.g., parallel algorithm development, workload partitioning, communications scheduling, etc). However, there are challenges that are specific to the distributed computing environment. This paper explores two of these challenges: the performance implications and policy issues associated with executing parallel programs on shared resources. We present accurate performance models that take into account both the mapping of application tasks to processors and the background load resulting from other users of the processors (e.g., the workstation's owner).

We focus on the important class of synchronous iterative (or multiphase) algorithms. This is a large class of algorithms, including optimization, discrete-event simulation, solution to sets of partial differential equations, Gaussian elimination, and many others. Several authors have derived performance models (and can find completion times) for synchronous iterative algorithms running on dedicated, homogeneous resources [4, 9, 13]. In contrast, analytic performance results for the use of shared, heterogeneous resources have been sparse. In this paper, we develop a performance modelling methodology for synchronous iterative algorithms executing on shared, heterogeneous resources. The performance model focuses on computational requirements; we assume a compute-intensive parallel application in which the computational requirements dominate. Hence, the modelling methodology is most accurate for relatively small processor populations (typically with less than 100 machines).

Once we have an accurate performance model, it can be used to help address a number of interesting policy questions that relate specifically to the use of shared computing resources. Essentially, we need to balance the competing requirements of the various users of the workstations. For an individual workstation's owner, the primary purpose of the machine is to service his or her day-to-day computing requirements (e.g., word processing, e-mail, etc). For these types of computing needs, minimum response time is typically of primary importance. For the initiator of a large parallel application, minimum completion time is also an important goal. However, for these two types of users to effectively share a common

† E-mail: gdp@el.wpafb.af.mil
‡ E-mail: roger@ccrc.wustl.edu

0967-1846/96/010009+11$19.50

machine (or set of machines), resource utilization policies must balance their competing interests.

The performance models can be used to effectively allocate computing resources under a variety of usage policies. Essentially, a policy is embodied into a cost function that can be optimized under a given set of constraints. The optimization problem is then solved to determine resource allocations. For example, if one wished to give complete priority to large parallel applications, the cost function to optimize might be the execution time of the parallel program. The optimization process could then choose the appropriate set of processors to minimize the completion time of the parallel application, independent of the impact the parallel execution has on other workstation users. At the other end of the policy spectrum, priority could be bestowed upon the workstation owner, and workstations that are not completely idle be barred from executing parallel jobs (this latter policy is used by the Condor [15] system).

A spectrum of policies exists between these two extremes. Such policies may reflect the relative cost of processing time on various machines, biases for or against the use of certain machines, and the relative importance of a workstation owner's load to the parallel application. By carefully constructing the cost function used in the optimization, different resource utilization policies can be evaluated. This yields near-optimal allocations of parallel tasks across the workstations while minimizing the impact on the original users of the workstations.

In the following section, we develop a performance model for synchronous iterative algorithms executing on shared, heterogeneous computing platforms. After validating the accuracy of the model, we then explore mechanisms for effectively utilizing these computing platforms under a spectrum of policy options. We show how changing the relative priority given to workstation owners and parallel applications can impact on the optimal resource allocation. Section 4 concludes and describes future work.

2. Performance model

Synchronous iterative algorithms repeatedly execute a computation, with an explicit synchronization of the tasks and exchange of data performed at the end of each computation (iteration). Each processor reaches a barrier synchronization after every iteration and awaits the arrival of the other processors before continuing. The model quantifies the performance effects of *application load imbalance*, the variance in runtime at each processor caused by an uneven distribution of the computation among the processors, and *background load*, the performance degradation resulting from other users of the shared resources. The model also accurately characterizes the performance of applications running on heterogeneous workstations with different computational speeds but similar architectures.

We employ two applications to illustrate and validate the performance models: a nonlinear optimization code and a globally-clocked, discrete-event simulation code. Biostatistics researchers employ the nonlinear optimization code in genetic epidemiology for determining whether genetic causes are responsible for diseases [7]. At its core, it minimizes a user provided cost function using quasi-Newton optimization methods. Essentially, a gradient descent is followed to the minimum of the cost function. Discrete-event queueing network simulation is used in the performance analysis of computer and communications systems. Here, a globally-clocked algorithm is utilized to simulate networks of FCFS queues. The job service requirements are exponentially distributed with departing jobs routed uniformly to neighbouring queues. These two codes are typical synchronous iterative applications.

At a high level, the runtime of synchronous iterative algorithms can be described by a simple performance model. In the ith iteration, a parallel algorithm requires some amount of time to complete serial calculations (operations that are not or cannot be parallelized) denoted $t_{serial,i}$. Similarly, each processor j completes some portion of the parallel computations for iteration i, requiring $t_{parallel,i,j}$. Because processors completing early must sit idle at the end of the iteration before the barrier synchronization operation, $\max_{1\le j\le P}(t_{parallel,i,j})$ gives the time required for the last processor to complete iteration i. Finally, the use of parallel processing typically results in some additional overhead, denoted $t_{par\_overhead,i}$. We include operations such as the barrier synchronization at the end of each iteration in $t_{par\_overhead,i}$. With I iterations of the algorithm, the runtime with P processors, R_P, can be modelled as:

$$R_P = \sum_{i=1}^{I}\left[t_{serial,i} + \max_{1\le j\le P} t_{parallel,i,j} + t_{par\_overhead,i}\right]. \tag{1}$$

For the applications in this paper, we assume each iteration requires roughly the same amount of computation. Therefore, we consider the computations required for a 'typical' iteration. To do so, we remove the direct reference to individual iterations in (1). We model the time needed to complete the serial and parallel overhead tasks in a typical iteration by the random variables $t_{serial,i}$ and $t_{par\_overhead,i}$. We define t_{serial} as the expected value of $t_{serial,i}$ and $t_{par\_overhead}$ as the expected value of $t_{par\_overhead,i}$. We model the time to complete the parallel tasks assigned to processor j in a typical iteration by the random variable $t_{parallel,j}$. The expectation of the maximum $t_{parallel,j}$ $(1 \le j \le P)$ describes the mean time required for the last processor to complete its parallel computations. Assuming the $t_{serial,i}$, $t_{par\_overhead,i}$, and the $t_{parallel,i,j}$ random variables are independent and identically distributed (iid), the runtime can be modelled as follows:

$$\begin{aligned} R_P &= \sum_{i=1}^{I}\Bigg(E\left[t_{serial,i}\right] + E\left[\max_{1\le j\le P} t_{parallel,i,j}\right] \\ &\qquad + E\left[t_{par\_overhead,i}\right]\Bigg) \\ &= I\left(t_{serial} + E\left[\max_{1\le j\le P} t_{parallel,j}\right] + t_{par\_overhead}\right). \end{aligned} \tag{2}$$

It is often beneficial to rewrite the parallel task completion time within an iteration in terms of the total workload across the processors (denoted by t_{PAR_WK}) rather than the maximum.

$$t_{PAR\_WK} = E\left[\sum_{j=1}^{P} t_{parallel,j}\right] = P \cdot E\left[t_{parallel,j}\right]. \quad (3)$$

Uneven distribution of the computation, *application load imbalance*, will impact on performance. Similarly, the tasks of other users of the shared resources compete with the distributed application. We refer to this as *background load imbalance*. To model the effects of application and background load imbalance, the expected value of the maximum task completion time is expressed as the average task completion time within an iteration multiplied by a load imbalance factor η.

$$E\left[\max_{1\le j\le P} t_{parallel,j}\right] = E[t_{parallel,j}] \cdot \eta = \frac{\eta\, t_{PAR\_WK}}{P}. \quad (4)$$

The resulting run time expression is:

$$R_P = I \cdot \left(t_{serial} + \frac{\eta t_{PAR\_WK}}{P} + t_{par\_overhead}\right). \quad (5)$$

Note that the ideal value of η is 1, which corresponds to a system with a single processor or a perfectly balanced system (i.e., no variation in $t_{parallel,j}$, and the expected value of the maximum equals the mean value). As the load imbalance worsens, η increases. Determining the value of η is one of the challenging aspects of developing an accurate performance model.

We assume the background load on each processor is independent and the parallel application load and background load are independent. We can model the total load imbalance on each processor j with the scale factor $\eta_j = \beta_j\gamma_j$, where the factors β_j and γ_j represent *application* and *background load imbalance*, respectively. These are combined with the normalization factor B (explained below) to find the maximum imbalance over all the processors, η:

$$\eta = \frac{1}{B} E\left[\max_{1\le j\le P}(\eta_j)\right] = \frac{1}{B} E\left[\max_{1\le j\le P}(\beta_j\ \gamma_j)\right]. \quad (6)$$

We define γ_j to be a discrete, positive integer-valued random variable. Similarly, we define β_j to be a positive, integer-valued random variable representing the amount of work done on processor j (in units of work) and introduce the normalization factor B, the average work for a processor. (Thus β_j/B is the application load imbalance scale factor for processor j.) Real valued β_j and γ_j are considered in [19]. The probability distribution for η_i can be expressed as follows:

$$Prob\left\{\eta_j = \beta_j\gamma_j = k\right\} = \sum_{\alpha=1}^{k} Prob\left\{\gamma_j = \alpha\right\} Prob\left\{\beta_j = \frac{k}{\alpha}\right\} \quad (7)$$

where $Prob\{\beta_j = k/\alpha\} = 0$ if k/α is not a positive integer. To model heterogeneous processors, we introduce the integer constants δ_j, the processing time per unit work of processor j, and Δ, the time per unit work of a baseline processor. For homogeneous resources, $\delta_j = \Delta = 1$. η is then redefined for heterogeneous processors as:

$$\eta = \frac{1}{\Delta B} E\left[\max_{1\le j\le P}(\eta_j)\right] = \frac{1}{\Delta B} E\left[\max_{1\le j\le P}(\beta_j\ \gamma_j\ \delta_j)\right]. \quad (8)$$

We can find the cumulative distribution and probability distribution of the individual processor load imbalance scale factors as follows:

$$Prob\left\{\max_{1\le j\le P}(\eta_j) \le k\right\} = \prod_{j=1}^{P}\sum_{\alpha=1}^{k} Prob\left\{\gamma_j = \alpha\right\} Prob\left\{\beta_j \le \left\lfloor \frac{k}{\alpha\delta_j} \right\rfloor\right\} \quad (9)$$

$$Prob\left\{\max_{1\le j\le P}(\eta_j) = k\right\} = Prob\left\{\max_{1\le j\le P}(\eta_j) \le k\right\} - Prob\left\{\max_{1\le j\le P}(\eta_j) \le k-1\right\} \quad (10)$$

with the floor present since β_j is integer-valued. Taking the normalizing factors B and Δ into account and finding the expectation yields:

$$\eta = \frac{1}{B\Delta}\sum_{k=1}^{\infty} k \times\left(\prod_{j=1}^{P}\sum_{\alpha=1}^{k} Prob\left\{\gamma_j = \alpha\right\} Prob\left\{\beta_j \le \left\lfloor \frac{k}{\alpha\delta_j} \right\rfloor\right\} - \prod_{j=1}^{P}\sum_{\alpha=1}^{k-1} Prob\left\{\gamma_j = \alpha\right\} Prob\left\{\beta_j \le \left\lfloor \frac{k-1}{\alpha\delta_j} \right\rfloor\right\}\right). \quad (11)$$

This model is widely applicable and has been used for the performance evaluation of a number of synchronous iterative applications. In this paper, the model is used to describe the performance of discrete-event simulation and nonlinear optimization [7, 20]. The general model has also been applied to Gaussian elimination and matrix multiplication [6]. A similar model is described by Atallah *et al* [1]; however, they assume a known constant background load and do not consider application load imbalance.

Having formulated a general performance model, we next investigate the distributions of γ_j and β_j. We first consider the impact of application load imbalance alone. We then consider background load imbalance and the effects of heterogeneity. Finally, we consider the interaction of application and background load imbalance on heterogeneous resources.

2.1. Application load imbalance

To refine the model to reflect the effects of application load imbalance alone, we begin by considering the discrete-event

simulation application running on dedicated, homogeneous resources. Here,

$$E[\max_{1\le j\le P} t_{parallel,j}] = \frac{\eta t_{PAR\_WK}}{P} = \frac{\eta N_e t_e}{P} \qquad (12)$$

where N_e is the mean number of events that are processed (on all processors) each iteration. Each event takes t_e time to be processed.

Three techniques for determining η are empirical measurement [5], modelling via order statistics [17], and analytic modelling via a known probability distribution [20]. We use the latter method. Starting from (11), we set $\gamma_j = 1$ to reflect no background load and $\delta_j = \Delta = 1$ to reflect the same computational speed across the processors.

$$\eta = \frac{1}{B}\sum_{k=1}^{\infty} k\left(\prod_{j=1}^{P} Prob\left\{\beta_j \le k\right\} - \prod_{j=1}^{P} Prob\left\{\beta_j \le k-1\right\}\right). \qquad (13)$$

Note that the normalization factor B, the average work for a processor, is simply $B = N_e/P$.

Each iteration, for each processor, we model the generation of an event via a Bernoulli trial for each of the simulated objects. If processor j has S_j simulated objects, each with a probability p to have an event at any time, then [20]:

$$Prob\left\{\beta_j \le \alpha\right\} = \sum_{l=0}^{\alpha}\binom{S_j}{l} p^l (1-p)^{S_j-l}. \qquad (14)$$

The average number of events per iteration, N_e can be found in a similar manner [20]. Each processor has a finite amount of work possible, so we define $\phi = \max_{1\le j\le P}(S_j)$, the maximum possible work for any processor. Then for dedicated, homogeneous processors

$$\eta = \frac{1}{B}\sum_{k=1}^{\phi} k\left(\prod_{j=1}^{P}\sum_{l=0}^{min(k,S_j)}\binom{S_j}{l} p^l (1-p)^{S_j-l} - \prod_{j=1}^{P}\sum_{l=0}^{min(k-1,S_j)}\binom{S_j}{l} p^l (1-p)^{S_j-l}\right). \qquad (15)$$

Using an approximation to make evaluation of η more tractable, we apply the DeMoivre–Laplace theorem [18]: if $S_j p(1-p) \gg 1$ then

$$\binom{S_j}{l} p^l (1-p)^{S_j-l} \approx \frac{1}{\sqrt{2\pi S_j p(1-p)}} \times \exp\left(-\frac{(l-S_j p)^2}{2S_j p(1-p)}\right). \qquad (16)$$

Therefore, we approximate η as follows:

$$\eta \approx \frac{1}{B}\sum_{k=1}^{\phi} k\left(\prod_{j=1}^{P}\sum_{l=0}^{min(k,S_j)} \frac{1}{\sqrt{2\pi S_j p(1-p)}} \times \exp\left(-\frac{(l-S_j p)^2}{2S_j p(1-p)}\right) - \prod_{j=1}^{P}\sum_{l=0}^{min(k-1,S_j)} \frac{1}{\sqrt{2\pi S_j p(1-p)}} \times \exp\left(-\frac{(l-S_j p)^2}{2S_j p(1-p)}\right)\right). \qquad (17)$$

If we assume that each processor has S simulated objects, then

$$\eta \approx \frac{(2\pi Sp(1-p))^{-\frac{P}{2}}}{B} \times \sum_{k=1}^{S} k\left[\left(\sum_{l=0}^{k} \exp\left(-\frac{(l-Sp)^2}{2Sp(1-p)}\right)\right)^P - \left(\sum_{l=0}^{k-1} \exp\left(-\frac{(l-Sp)^2}{2Sp(1-p)}\right)\right)^P\right]. \qquad (18)$$

We now have an analytic expression for application load imbalance for a discrete-event simulation application executing on dedicated, homogeneous resources.

2.2. Imbalance due to background load

We next turn to modelling the imbalance due to background load and focus on genetic epidemiological optimization applications without any appreciable application load imbalance (i.e., $\beta_j = 1$). We assume the application and the background load are independent, the background load on each processor is independent and identically distributed, all tasks on a processor have the same priority, and (initially) the processors are homogeneous. Substituting $\beta_j = B = 1$ and $\delta_j = \Delta = 1$ into (11) yields

$$\eta = \sum_{k=1}^{\infty} k\left(\prod_{j=1}^{P} Prob\left\{\gamma_j \le k\right\} - \prod_{j=1}^{P} Prob\left\{\gamma_j \le k-1\right\}\right). \qquad (19)$$

If the background load on processor j is k tasks, and each task (including the application) gets an equal fraction of the CPU time, the application will take $k+1$ times as long as if it were run on an idle processor. Hence, $\gamma_j = k+1$ in this case. Assuming the existence of a background load distribution $Q_{j,k}$ (the probability of k background tasks on processor j), $Prob\left\{\gamma_j = k+1\right\} = Q_{j,k}$. This distribution can be derived via a queueing model of the background workload. We assume that the background jobs arrive following a Poisson process with a given rate λ_j and consider two queueing models to describe the background load. The first, a Processor Sharing (PS) model, is an idealization of round robin scheduling where the time quantum for each process approaches zero [12]. Assuming that the service distribution is Coxian [11], the queue length distribution for a server is $Q_{j,k} = (1-\rho_j)\rho_j^k$, where ρ_j is the ratio of the arrival rate to the service rate ($\rho_j = \lambda_j/\mu_j$). For general service distributions, finding the queue length distribution for a processor sharing server is impractical.

For the processor sharing case with Coxian distributed service times,

$$Prob\{\gamma_i \le \alpha + 1\} = \sum_{l=0}^{\alpha}(1-\rho_j)\rho_j^l$$
$$= (1-\rho_j)\left(\frac{1-\rho_j^{\alpha+1}}{1-\rho_j}\right) = 1-\rho_j^{\alpha+1}$$
$$Prob\{\gamma_j \le \alpha\} = 1-\rho_j^{\alpha}. \quad (20)$$

Assuming processor sharing on shared, homogeneous resources running the nonlinear optimization application, the load imbalance is given by:

$$\eta = \sum_{k=1}^{\infty} k\left[\prod_{j=1}^{P}\left(1-\rho_j^k\right) - \prod_{j=1}^{P}\left(1-\rho_j^{k-1}\right)\right]. \quad (21)$$

To extend the processor sharing model to heterogeneous resources, we substitute $\beta_j = B = 1$ into (11).

$$\eta = \frac{1}{\Delta}\sum_{k=1}^{\infty} k\left[\prod_{j=1}^{P} Prob\left\{\gamma_j \le \left\lfloor \frac{k}{\delta_j}\right\rfloor\right\} - \prod_{j=1}^{P} Prob\left\{\gamma_j \le \left\lfloor \frac{k-1}{\delta_j}\right\rfloor\right\}\right]. \quad (22)$$

We also include the impact of heterogeneity on the service rate for background tasks by defining the service rate $\mu_j = \mu\ \Delta/\delta_j$, where μ_j is simply the baseline service rate μ scaled to reflect the relative processing power of processor j. Applying this result into (20) yields:

$$Prob\left\{\gamma_j \le \left\lfloor \frac{k}{\delta_j}\right\rfloor\right\} = 1-\left(\frac{\delta_j\lambda_j}{\mu\Delta}\right)^{\left\lfloor \frac{k}{\delta_j}\right\rfloor}. \quad (23)$$

The load imbalance for the optimization application running on shared, heterogeneous resources and assuming processor sharing is then

$$\eta = \frac{1}{\Delta}\sum_{k=1}^{\infty} k\left[\prod_{j=1}^{P}\left(1-\left(\frac{\delta_j\lambda_j}{\mu\Delta}\right)^{\lfloor k/\delta_j\rfloor}\right) - \prod_{j=1}^{P}\left(1-\left(\frac{\delta_j\lambda_j}{\mu\Delta}\right)^{\lfloor (k-1)/\delta_j\rfloor}\right)\right]. \quad (24)$$

A second queueing model, based on FCFS M/G/1 queueing theory, is also used to derive the queue length distributions, allowing us to include more general service distributions. Given a service time density, $b_j(x)$, and its Laplace–Stiltjes transform $B_j^*(s)$, the Pollaczek–Khinchine transform formula [12] is employed to find the z-transform of the queue length distribution for processor j.

$$Q_j(z) = B_j^*(\lambda_j - \lambda_j\ z)\frac{(1-\rho_j)(1-z)}{B_j^*(\lambda_j - \lambda_j\ z) - z}. \quad (25)$$

The queue length distribution is then found by performing an inverse z-transform.

To understand the service requirements of typical UNIX processes, Leland and Ott collected statistics on 9.5 million processes [14]. They found that exponential distributions are not accurate for modelling service times, but a good model for the cumulative distribution function of service time is:

$$F(x) \approx 1 - rx^{-c} 1.05 < c < 1.25 \quad (26)$$

with the best results with $r = 0.241$ and $c = 1.122$. This model is only accurate for jobs of at least 1 second duration, and does not have a finite variance. Order statistics (such as the expected maximum of P random variables) which are drawn from distributions with infinite variance are unbounded [8]. To address this, we have modified this to a density function as follows:

$$f(x) = \begin{cases} 0.759 & \text{if } 0 \le x < 1 \\ crx^{-(c+1)} & \text{if } 1 \le x \le 1000 \\ 0 & \text{if } x > 1000 \end{cases} \quad (27)$$

Although this distribution now has a finite variance, the Pollaczek–Khinchine transform yields a solution whose z-transform inversion is intractable. Therefore, we do not use this distribution in the performance model, but we will use synthetic workloads in accordance with this distribution to validate the modelling methodology.

In their simulations of load balancing algorithms, Banawan and Zeidat utilize a hyperexponential (mixed exponential) service distribution to model the required CPU time of processes [2]. Based on these results, we model the service distribution as a hyperexponential distribution with two types of customers. For some probabilities $\pi_{a,j}$ and $\pi_{b,j} = 1 - \pi_{a,j}$ and service rates $\mu_{a,j}$ and $\mu_{b,j}$, we have the density

$$b_j(x) = \pi_{a,j}\ \mu_{a,j}\ \mathrm{e}^{-x\mu_{a,j}} + \pi_{b,j}\ \mu_{b,j}\ \mathrm{e}^{-x\mu_{b,j}} \quad (28)$$

and

$$B_j^*(s) = \frac{\pi_{a,j}\ \mu_{a,j}}{\mu_{a,j}+s} + \frac{\pi_{b,j}\ \mu_{b,j}}{\mu_{b,j}+s}. \quad (29)$$

System accounting tools were used to collect statistics on the service requirements of processes on numerous machines for several weeks. Similarly, arrival rate statistics for the machines were accumulated. These statistics are used to determine the service and arrival rate distributions in the model. We choose the values of λ_j, the arrival rate of jobs at processor j; $\pi_{a,j}$ and $\pi_{b,j}$, the probabilities of each type job; and $\mu_{a,j}$ and $\mu_{b,j}$, the service rates of each type of job based on the empirical results.

2.3. Application and background load imbalance

Having developed performance models for both application load imbalance and background load imbalance, we now scrutinize their combined effects. Once again, discrete-event simulation is the parallel application of interest.

We now model the load imbalance on shared heterogeneous resources assuming γ_j and β_j are independent discrete random variables,

$$Prob\{\eta_j \le k\} = Prob\{\gamma_j\beta_j \le \lfloor k/\delta_j\rfloor\}$$
$$= \sum_{\alpha=1}^{k} Prob\{\beta_j = \alpha\}\ Prob\left\{\gamma_j \le \left\lfloor \frac{k}{\alpha\delta_j}\right\rfloor\right\}. \quad (30)$$

If we restrict ourselves to processor sharing, then from the derivation in section 2.2,

$$Prob\{\eta_j \le k\} = \sum_{\alpha=1}^{k} Prob\{\beta_j = \alpha\} \times \left(1 - \left(\frac{\Delta\lambda_j}{\mu\delta_j}\right)^{\lfloor k/\alpha\delta_j \rfloor}\right). \quad (31)$$

The load imbalance is then

$$\eta = \frac{1}{B\Delta}\sum_{k=1}^{\infty} k\left(\prod_{j=1}^{P} Prob\{\eta_j \le k\} - \prod_{j=1}^{P} Prob\{\eta_j \le k-1\}\right) \quad (32)$$

$$= \frac{1}{B\Delta}\sum_{k=1}^{\infty} k\left(\prod_{j=1}^{P}\sum_{\alpha=1}^{k} Prob\{\beta_j = \alpha\} \times \left(1 - \left(\frac{\Delta\lambda_j}{\mu\delta_j}\right)^{\lfloor k/\alpha\delta_j \rfloor}\right) - \prod_{j=1}^{P}\sum_{\alpha=1}^{k-1} Prob\{\beta_j = \alpha\} \times \left(1 - \left(\frac{\Delta\lambda_j}{\mu\delta_j}\right)^{\lfloor (k-1)/\alpha\delta_j \rfloor}\right)\right). \quad (33)$$

We apply the results of the Bernoulli trials and the DeMoivre–Laplace theorem to find η for the simulation application. The load imbalance is given by

$$\eta = \frac{1}{\Delta B}\sum_{k=1}^{\infty} k\left(\prod_{j=1}^{P}\sum_{\alpha=1}^{\min(S_j\delta_j,k)} \frac{1}{\sqrt{2\pi S_j p(1-p)}} \times \exp\left(-\frac{(\alpha - S_j p)^2}{2S_j p(1-p)}\right)\left(1 - \left(\frac{\Delta\lambda_j}{\mu\delta_j}\right)^{\lfloor k/\delta_j\alpha \rfloor}\right) - \prod_{j=1}^{P}\sum_{\alpha=1}^{\min(S_j\delta_j,k-1)} \frac{1}{\sqrt{2\pi S_j p(1-p)}} \times \exp\left(\frac{-(\alpha - S_j p)^2}{2S_j p(1-p)}\right)\left(1 - \left(\frac{\Delta\lambda_j}{\mu\delta_j}\right)^{\left\lfloor \frac{k-1}{\delta_j\alpha} \right\rfloor}\right)\right). \quad (34)$$

Using this equation and the general performance model given in (5), we can describe the performance of the discrete-event simulation application running on shared, heterogeneous resources.

2.4. Model validation

We first consider the impact of background load on the nonlinear optimization application. Using synthetic loading corresponding to the results of Leland and Ott discussed above, we varied the background load by increasing the arrival rate of background tasks λ. As shown in figure 1, the model accurately predicts the impact of background load imbalance.

Experiments with moderately loaded, heterogeneous workstations were also performed. Figure 2 shows the predicted and measured performance of the nonlinear optimization application on a network of workstations. The poor performance with two processors is caused by the fact that, although the second processor in the pool is more powerful than the other processors, it is heavily loaded. The performance model accurately predicted the performance of the nonlinear optimization executing on the shared, heterogeneous workstations.

The discrete-event simulation application was executed on an nCUBE/7 hypercube to validate the model when there is application load imbalance, but no background load imbalance [20]. Figure 3 shows that the model is

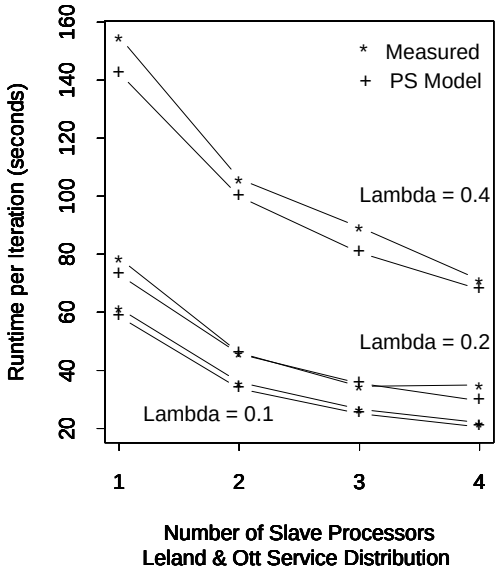

Figure 1. Nonlinear optimization on shared, homogeneous resources.

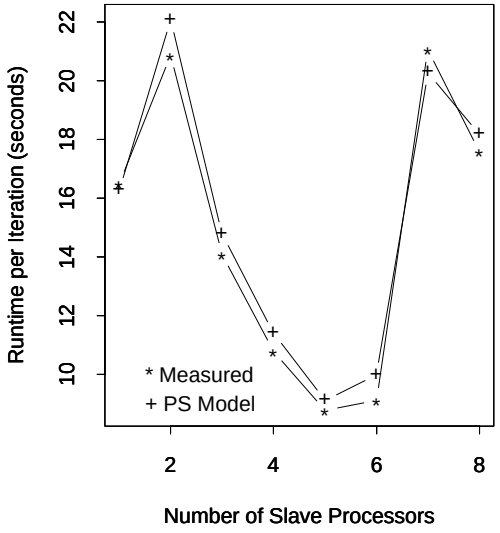

Figure 2. Nonlinear optimization on shared, heterogeneous resources.

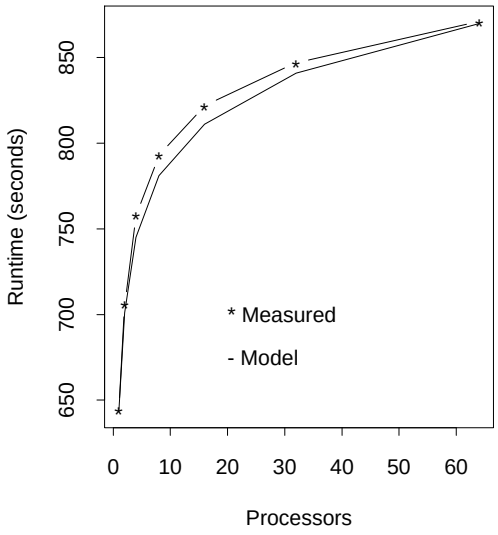

Figure 3. Simulation on dedicated, homogeneous resources.

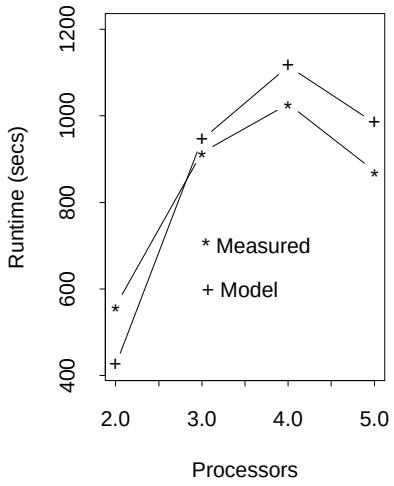

Figure 4. Simulation on shared, heterogeneous resources.

accurate in this instance also. Finally, the discrete-event simulation application was executed on a network of shared workstations to validate the model when there is both application and background load imbalance [21]. Although the model error is larger in this instance, the model is still reasonably accurate, reliably reflecting measured performance trends, as illustrated in figure 4.

3. Usage policies and parallel application scheduling

When two or more parties are sharing a common resource, the effective utilization of that resource must be measured against a desired policy, or usage goal. Here, we are interested in investigating various policies to guide the sharing of a network of workstations between the individual workstation owners and distributed applications. We are also interested in mechanisms to carry out those policies. Specifically, we help to solve the following problem: given a set of workstations with some existing background load pattern and a parallel application with known computational requirements, choose an appropriate set of workstations on which to execute the parallel application that best meets a set of predetermined policy goals.

A number of issues must be addressed when establishing a usage policy. Defining the relative importance of parallel applications compared to background load is one important policy issue. In addition, it is often desirable to differentiate between the individual workstations in the pool (i.e., guiding the parallel application towards some machines and away from others) due to their processing power, current workload, or other external factors.

To implement a desired policy, we will incorporate the goals of that policy into a cost function that rates how well the assignment of parallel tasks to workstations meets the policy goals. By using optimization techniques to minimize the cost function, the parallel application can be scheduled to most efficiently utilize the shared computational resources.

Optimization of a general cost function is an NP-hard problem. Therefore, true optimization is often limited to restricted classes of problems which have efficient solutions. A number of restricted scheduling problems have been studied with efficient algorithms for finding optimal solutions described in [3, 23, 25]. Atallah *et al* investigate parallel algorithms similar to those considered in this paper and give an algorithm for finding the optimal set of homogeneous processors to minimize runtime assuming a known constant background load [1]. To determine the schedule with the minimum parallel application runtime we use the algorithm shown in figure 5, where the sort is based on the arrival rate of background jobs. This algorithm, similar to the one described in [1], gives the optimal solution.

For parallel applications running on heterogeneous resources, the performance model and cost function are significantly more complex. To find the minimum runtime in this case, we resort to a *greedy* heuristic to find a near-optimal solution [16, 22]. If we assume that the parallel application is divided equally among each of the processors, then we only need to consider the background load and heterogeneity to determine the set of processors to use. The background load is modelled with a processor sharing queueing model, so the expected number of background tasks at processor j is $\rho_j/(1-\rho_j)$, assuming that the service distribution of background tasks is Coxian. Adding the parallel application, the expected number of tasks at

```
Sort available processors S
P ←{i}, where i is processor in S with lowest arrival rate
while not done do
                    P' ←P ∪ {j}, where j is next processor in S
                    if R_P' < R_P
                        P ←P'
                    else
                        done ← TRUE
endwhile
return optimal set P
```

Figure 5. Optimal algorithm for minimum runtime (homogeneous processors).

processor j is $1/(1-\rho_j)$. Because the processors are heterogeneous, we scale the runtime by δ_j/Δ. Multiplying the background load and heterogeneity scale factors, we find that processor j is expected to take $\delta_j/[\Delta(1-\rho_j)]$ times as long as if run on an idle baseline processor. For the heuristic, we sort the processors based on the values of this scale factor, and find a near-optimal set of processors P using the same algorithm as shown in figure 5.

The above two algorithms (for homogeneous and heterogeneous processors) implement a policy that ignores the impact of the parallel application on the other workstation users. By minimizing the execution time of the parallel application, without considering the impact on other users, the cost function essentially gives the parallel application priority over the owners of the workstations.

We next introduce a more flexible cost function that explicitly reflects the importance of minimizing the completion time of the parallel application as well as minimizing the impact that the parallel application has on other workstation users. We assume that there is some cost per unit time, x, which reflects the goal of minimizing the runtime of the parallel application. If each processor j used in the computation has some cost per unit time, c_j, associated with its usage by the application, then we formulate a cost function, C_P, which reflects the trade-off between maximizing application performance and the expense of using the workstations:

$$C_P = xR_P + \sum_{j=1}^{P} c_j R_P = R_P\left(x + \sum_{j=1}^{P} c_j\right). \quad (35)$$

By assigning appropriate c_j values, the cost function reflects policies concerning which workstations are encouraged for use with parallel applications. As the c_j values increase, the cost of using additional processors increases, making the use of fewer processors more likely to be optimal. In contrast to the individual processor costs, x is used to reflect the importance of minimizing the execution time of the application, regardless of the number of processors. As we change the relative values of x and the c_j terms, we have a spectrum of policies that describe the relative importance of application performance and its impact on background processing.

We explore one example of this type of policy by considering a network of homogeneous workstations with identical costs for the use of each workstation (in this case, the cost function is $C_P = R_P(x + cP)$). Once again we find the optimal solution using the algorithm in figure 5, replacing R_P with C_P. Because the algorithm orders the processors and adds processors to P in order,

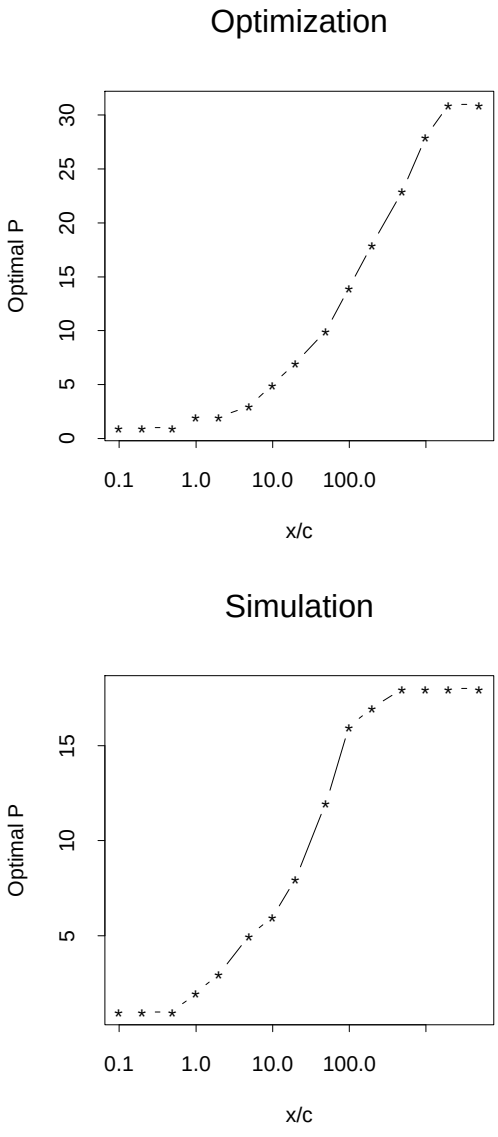

Figure 6. Optimal set of homogeneous processors.

knowing the number of processors in P is sufficient information to describe which processors are in P. Figure 6 displays the optimal number of processors to use for the nonlinear optimization and discrete-event simulation applications running on a homogeneous network of 32 workstations. In this figure, we vary the ratio x/c to reflect a variation in the relative importance of the parallel application and the workstation owner's load. With enough priority given to the workstations' owners (low x/c), the parallel application is assigned to only one processor. With enough priority given to the parallel application (high x/c), the cost function effectively minimizes the application runtime. Between these two extremes, the number of processors assigned to the parallel application reflects the assigned importance of the application and the background workload at each processor.

When running the application on heterogeneous processors, we need to use a heuristic as before, again replacing R_P with C_P. In this example, the heterogeneous network contains 32 workstations of four different speeds, as denoted by the value δ_j for processor j. Six of the processors have $\delta_j = 1$, six have $\delta_j = 2$, ten have $\delta_j = 4$ and ten have $\delta_j = 8$. The performance of the serial case is found by using the fastest processor while unloaded.

The near-optimal number of processors to use for the two applications when running on heterogeneous workstations is shown in figure 7. As in the homogeneous case, we vary the relative importance of the parallel application and the background load. Again, when the parallel application is given low priority (low x/c), it is assigned to only a single processor (in this case, the fastest processor with the lowest background load). When the parallel application is given higher priority (high x/c), a larger set of processors is used.

The previous two examples assigned an equal priority to each of the available workstations (i.e., $\forall j, c_j = c$). By assigning different values of c_j to different workstations, a number of different policies can be reflected. In the simplest case, assigning a higher c_j to some workstations and a lower c_j to others discourages the use of the workstations with higher c_j. For example, the workstation where the parallel application is initiated could have a low c_j to encourage the application to run locally and only branch out to other workstations if there is a significant runtime benefit to doing so. Additional sophistication can be incorporated into the usage policies by making the c_j values functions of either time of day and/or current load. As functions of the time of day, parallel applications can be encouraged to use a larger set of workstations when the workstations' owners are not expected to be present. By making c_j a function of the current workload at processor j, the policy can further discourage the use of already loaded workstations.

So far we have considered mechanisms that assign varying priorities to the parallel application or to the other workstation load, but do not put hard limits on where (or for how long) the parallel application is allowed to execute. Policies that include such limits can be implemented using a constrained optimization.

For example, consider a policy that states the parallel application should be assigned to an appropriate set of processors so that it is expected to complete in a specified amount of time, but it is to be assigned to processors in such a way as to minimize the impact on other workstation load while still meeting its runtime goal. This policy could be formulated as the following optimization problem: given an execution time goal G_R, minimize

$$\sum_{j=1}^{P} c_j R_P$$

under the constraint $R_P \leq G_R$. Similarly, a policy that bounds the resource utilization of the parallel application and desires to minimize the execution time under a utilization constraint could be formulated as the following: given a resource utilization bound G_C, minimize R_P under

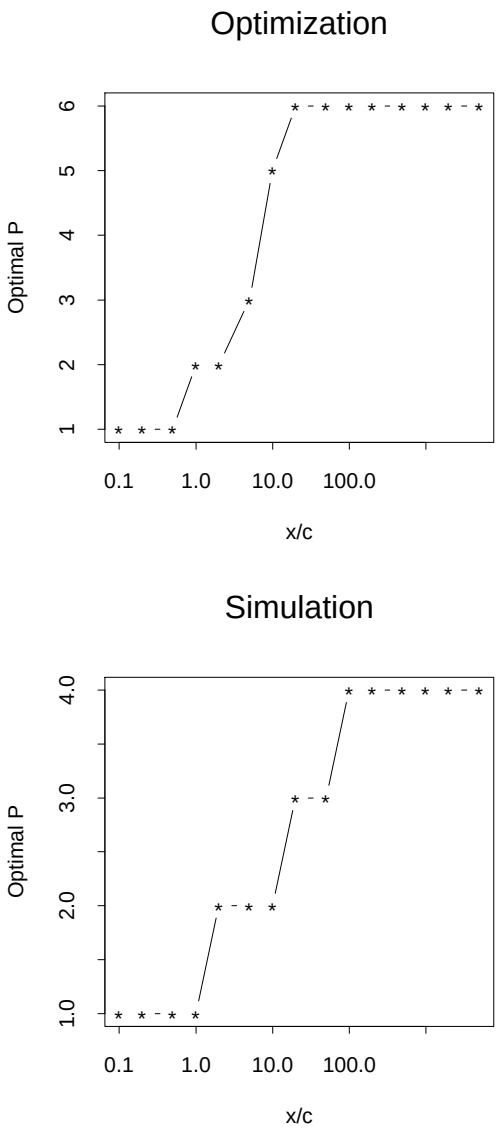

Figure 7. Near-optimal set of heterogeneous processors.

the constraint

$$\sum_{j=1}^{P} c_j R_P \leq G_C.$$

In both of the above two examples, we are essentially constraining one of the two terms in (35) and minimizing the other term.

As the cost function becomes more complicated (either by varying the c_j values or imposing additional constraints), the optimization problem becomes significantly more difficult. As a result, the simple algorithms described above are no longer effective and general optimization techniques must be used.

The above discussion shows how one can use performance models of parallel applications executing on networks of workstations to help implement usage policies that guide the sharing of available computational resources. A policy is embodied in a cost function that is minimized to yield the optimal (or near-optimal) assignment of processors for a parallel application execution. Depending on the form of the cost function to be optimized, either an optimal algorithm, heuristic, or general optimization technique is the most appropriate.

4. Conclusions and future work

There is a great desire to more efficiently utilize the computational resources that currently go unused on individuals' desks. The systems that support the use of these resources, however, are in their infancy, and therefore do not yet provide effective coordinated sharing of machine cycles. They typically fall at one end of the policy spectrum or the other, ignoring the impact they are having on a workstation's owner or checkpointing and moving somewhere else if they suspect that the owner is currently using a machine. We are exploring the middle ground, where computationally intensive applications are allowed to exploit underutilized desktop workstations while being cognizant of the impact they are having on the workstation's primary user.

To this end, we have developed performance models for synchronous iterative algorithms that accurately characterize parallel application performance in the presence of *application load imbalance*, *background load* due to other users, and *heterogeneity* across the processor set. These models are used to formulate a cost function that reflects the policy goals of the organization. This cost function is then optimized to guide the scheduling of the parallel application onto an appropriate set of processors. By an appropriate choice of cost function (and associated optimization method), a wide variety of policy choices can be supported.

Although the results presented here are positive (i.e., the performance models reflect actual runtimes, the cost functions characterize the policy goals, and the optimization algorithms yield viable parallel algorithm assignments), there are a number of factors limiting the widespread adoption of the techniques described here. These factors point to additional research that is needed. First, the relationship between the cost function and policy goals relies heavily on the accuracy of the performance model. Although an accurate performance model was presented for synchronous iterative algorithms, many parallel applications do not fit the model and would need an alternative performance predictor for effective use. Second, the development described in this paper assumed the problem of assigning a set of processors to execute only one parallel application. The methods developed here must be extended to support multiple parallel applications executing simultaneously. Efe and Schaar [10] have some results in this area, supporting co-scheduling of parallel applications. Third, once assigned to a processor, parallel tasks are assumed to be of equal priority to the other workstation load. We are currently extending the performance model to accurately reflect priority assignments on individual workstations. Fourth, the cost function optimization is likely to be a computationally intensive operation in its own right. Either the cost functions must be constrained so that the scheduling decisions can be made efficiently, or more efficient heuristics must be developed to find near-optimal solutions to sophisticated cost functions. In spite of the limitations described above, we are clearly making progress in supporting the effective sharing of currently underutilized resources.

References

[1] Atallah M J, Black C L, Marinescu D C, Siegel H J and Casavant T L 1992 Models and algorithms for coscheduling compute-intensive tasks on a network of workstations *J. Parallel Distrib. Comput.* **16** 319–27

[2] Banawan S A and Zeidat N M 1992 A comparative study of load sharing in heterogeneous multicomputer systems *Proc. 25th Ann. Simulation Symp. (Orlando, FL)* (Los Alamitos, CA: IEEE Computer Society Press) pp 22–31

[3] Bokhari S H 1988 Partitioning problems in parallel, pipelined, and distributed computing *IEEE Trans. Comput.* **37** 48–57

[4] Brochard L, Prost J-P and Faurie F 1989 Synchronization and load unbalance effects of parallel iterative algorithms *Proc. Int. Conf. on Parallel Processing (St Charles, IL)* vol 1 (University Park, PA: The Pennsylvania State University Press) pp 153–60

[5] Chamberlain R D and Franklin M A 1990 Hierarchical discrete-event simulation on hypercube architectures *IEEE Micro* **10** (4) 10–20

[6] Chamberlain R D and Franklin M A 1993 Performance effects of synchronization in parallel processors *Proc. 5th IEEE Symp. on Parallel and Distributed Processing (Dallas, TX)* (Los Alamitos, CA: IEEE Computer Society Press) pp 611–6

[7] Chamberlain R D, Franklin M A, Peterson G D and Province M A 1995 Genetic epidemiology, parallel algorithms, and workstation networks *Proc. 28th Hawaii Int. Conf. on System Sciences (Maui, HI)* (Los Alamitos, CA: IEEE Computer Society Press) pp 101–10

[8] David H A 1970 *Order Statistics* (New York: Wiley)

[9] Dubois M and Briggs F A 1982 Performance of synchronized iterative processes in multiprocessor systems *IEEE Trans. Software Engng* **SE-8** 419–31

[10] Efe K and Schaar M A 1993 Performance of co-scheduling on a network of workstations *Proc. 13th Int. Conf. on Distributed Computing Systems* pp 525–31

[11] Harrison P G and Patel N M 1993 *Performance Modelling of Communications Networks and Computer Architectures* (Reading, MA: Addison-Wesley)

[12] Kleinrock L 1975 *Queueing Systems* vol 1 and 2 (New York: Wiley)
[13] Kruskal C P and Weiss A 1985 Allocating independent subtasks on parallel processors *IEEE Trans. Software Engng* **SE-11** 1001–16
[14] Leland W and Ott T 1986 Load balancing heuristics and process behavior *Proc. PERFORMANCE '86 and ACM SIGMETRICS (Rayleigh, NC)* (New York: ACM Press) pp 54–69
[15] Litzkow M and Livny M 1990 Experience with the Condor distributed batch system *Proc. IEEE Workshop on Experimental Distributed Systems (Huntsville, AL)* (Los Alamitos, CA: IEEE Computer Society Press)
[16] Lo V M 1988 Heuristic algorithms for task assignment in distributed systems *IEEE Trans. Comput.* **37** 1384–97
[17] Madala S and Sinclair J B 1991 Performance of synchronous parallel algorithms with regular structures *IEEE Trans. Parallel Distrib. Syst.* **2** 105–16
[18] Papoulis A 1984 *Probability, Random Variables, and Stochastic Processes* (New York: McGraw Hill)
[19] Peterson G D 1994 Parallel application performance on shared, heterogeneous workstations *DSc Thesis* Washington University in St. Louis
[20] Peterson G D and Chamberlain R D 1994 Beyond execution time: expanding the use of performance models *IEEE Parallel Distrib. Technol.* **2** (2) 37–49
[21] Peterson G D and Chamberlain R D 1994 Sharing networked workstations: a performance model *Proc. 6th IEEE Symp. on Parallel and Distributed Processing (Dallas, TX)* (Los Alamitos, CA: IEEE Computer Society Press) pp 308–15
[22] Ramamritham K, Stankovic J A and Shiah P-F 1990 Efficient scheduling algorithms for real-time multiprocessor systems *IEEE Trans. Parallel Distrib. Syst.* **1** 184–94
[23] Stone H S 1978 Critical load factors in two-processor distributed systems *IEEE Trans. Software Engng* **SE-4** 254–8
[24] Sunderam V S 1990 A framework for parallel distributed computing *Concurrency: Practice Exper.* **2** 315–39
[25] Tantawi A N and Towsley D 1985 Optimal load balancing in distributed computer systems *J. ACM* **32** 445–65
[26] Turcotte L H 1993 A survey of software environments for exploiting networked computing reources *Technical Report MSU-EIRS-ERC-93-2* NSF Engineering Research Center for Computational Field Simulation, Mississippi State University, Starkville, MS