Hierarchical Job Scheduling for Clusters of Workstations

J. Santoso [1,2] G.D. van Albada [1] B.A.A. Nazief [3] P.M.A. Sloot [1]

[1] Section Computational Science, Univ. van Amsterdam
[2] Department of Informatics,Bandung Institute of Technology
Jl. Ganesha 10, Bandung 40132, Indonesia
[3] Department of Computer Science,University of Indonesia
Jakarta, Indonesia
{judhi, dick, sloot}@wins.uva.nl nazief@cs.ui.ac.id

Keywords: Cluster Computing, Scheduling

Abstract [1]

In this paper we study hierarchical job scheduling strategies for clusters of workstations. Our approach uses two-level scheduling: global scheduling and local scheduling. The local scheduler refines the scheduling decisions made by the global scheduler, taking into account the most recent information. In previous work, we explored the First Come First Served (FCFS), the Shortest Job First (SJF), and the First Fit (FF) policies at the global level. Now, we will apply all three policies both at the global and the local level. In addition, we use separate queues at the global level for arriving jobs, where the jobs with the same number of tasks are placed in one queue. At both levels, the schedulers strive to maintain a good load balance. The unit of load balancing at the global level is the job consisting of one or more parallel tasks; at the local level it is the task.

1 Introduction

With the advent of large computing power in workstations and high speed networks, the high-performance computing community is moving from the use of massively parallel processors (MPPs) to cost effective clusters of workstations. Furthermore, integrating a set of clusters of workstations into one large computing environment can improve the availability of computing power, e.g Globus (Czajkowski et al. [3]), the Polder Metacomputing Initiative [9]. However, to efficiently exploit the available resources, an appropriate resource management or scheduling system is needed. The scheduling system for large environments can be divided into two levels: scheduling across clusters (*wide area scheduling*) and scheduling within a cluster. The wide area scheduler queries the local scheduler to obtain the candidate resources to allocate the submitted jobs [11].

Scheduling systems within a single cluster of workstations have been described in the literature [1], [5], [7], [10], [12]. Various strategies are used to achieve the objective of scheduling. For example two-level time-sharing scheduling for parallel and sequential jobs has been introduced by Zhou et al. [12] to attain a good performance for parallel jobs and a short turn-around time for sequential jobs. At the upper level, time slots are used. Each time slot is divided into 2 slots (one for sequential jobs and a second for tasks of parallel jobs). However at the local level a task of a parallel job may share its time slot with sequential jobs. K.Y Wang[10] uses hierarchical decision scheduling that is implemented using global and local schedulers. The global scheduler is responsible for long term allocation of system resources, and the local scheduler is responsible for short term decisions concerning processor allocation.

In this paper we use two level scheduling in a different way: global scheduling across clusters and local scheduling within a cluster. The global scheduler has a number of functions. One of these is the matching of the resources requested by a job to those available in the participating clusters. Another is to obtain the best utilisation of the available clusters. The local scheduler is responsible for scheduling jobs to a specific resource. The global scheduler and local schedulers have various scheduling policies that can be used independently. To observe our scheduling strategies' performance, we have developed a simu-

[1]Paper appeared in: J. Santoso; G.D. van Albada; B.A.A. Nazief and P.M.A. Sloot: Hierarchical Job Scheduling for Clusters of Workstations, in L.J. van Vliet; J.W.J. Heijnsdijk; T. Kielmann and P.M.W. Knijnenburg, editors, ASCI 2000, Proceedings of the sixth annual conference of the Advanced School for Computing and Imaging, pp. 99-105. ASCI, Delft, June 2000. ISBN 90-803086-5-x.

lation environment. By using this environment, the influence of scheduling strategies on the over-all performance will be shown.

We are not so much be concerned with the architecture and implementation of the scheduling system, but rather with the scheduling strategies that can be used to attain the best possible performance in our environment. We will specifically address the prioritisation of parallel jobs at the global scheduling level. To support this, we explore the use of multiple queues at the global level.

This paper is organised as follows: In section 2 we discuss the the relevant aspects of our scheduling model as well as several scheduling policies that we will use. In section 3 we explain our simulation environment including job and resource descriptions. The simulation results are presented in section 4, after which we present our conclusions.

2 The scheduling model

The scheduling model is based on a hierarchical approach. We distinguish two levels of scheduling: scheduling at the top level (global scheduling) and scheduling at the local level. The global and local scheduler interact with each other to make an optimal schedule. The local scheduler will refine the scheduling decisions made by global scheduler to adapt to changes in resource availability.

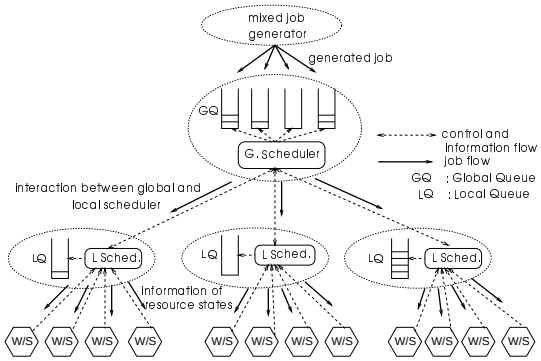

Figure 1: Global and Local Scheduler interaction

2.1 The global scheduler

The global scheduler schedules the arriving jobs to the available clusters. Once a job has been scheduled by the global scheduler, it is passed on to the local cluster scheduler. For low loads, the global scheduler's task is to find the best offer, in terms of e.g. performance or cost. Jobs are handled on an individual basis, and no global queue needs to be maintained. For high system loads (typical for high-throughput computing), the scheduler should also try to maximise the system utilisation and reduce the over-all turnaround time. To this end, the scheduler should know the available capacity in each participating cluster. As, due to locally submitted work, this capacity may vary in an unpredictable manner, it may be attractive to delay scheduling until the needed capacity actually becomes available. We assume that jobs delayed in this way are maintained in a global scheduling queue. We have studied two distinct queuing strategies at the global level: a single global queue and separate global queues. In the single queue model, one of three scheduling strategies is used to assign jobs to clusters: First Come First Served (FCFS), Shortest Job First (SJF) or First Fit (FF). The FF policy maintains a queue in FCFS order, but searches that queue for the first job that can be scheduled on the available resources. SJF and FCFS wait until the first job in the queue can be scheduled. FF should result in a good resource utilisation, but has a bias against large multi-task jobs as it can schedule sequential jobs ahead of them.

When using separate queues, each type of job (distinguished by the number of tasks), is placed in separate queue. The number of queues is equal to the number of job types. Within each queue the SJF policy is used. Dispatching starts either from the queue of jobs with the smallest number of tasks or, alternatively, from the queue with the largest number of tasks. We call the first strategy the S-SJF policy and the second strategy, L-SJF.

The global scheduler interacts with the local scheduler to make the best schedule. The interaction between the local and global scheduler depends on whether a local queue is used. When no local queue is used, the global scheduler will match the job's resource request to the currently available local resources. When a local queue is used, the maximum capacity of the cluster as well as the current and maximum queue length are considered.

2.2 The local scheduler

The role of the local scheduler is important for the overall system performance. The local scheduler tries to maximise the resource utilisation by using the most recent resource information in making decisions. The resource (workstation) with the largest capacity will be selected first to allocate jobs. One of three scheduling policies is used to select the jobs from the queue: FCFS, SJF or FF. The local scheduler uses only one queue for all types of job. The maximum local queue size is limited.

At this level we also implement task scheduling of parallel jobs. The local scheduler will schedule only a limited number of tasks on a specific node. Tasks are assigned to available nodes in a Round-Robin fashion starting from the most lightly loaded node.

The tasks on each node are scheduled in a Round-Robin fashion, unless they are blocked, waiting for e.g. synchronisation with the other tasks in the job.

3 The simulation environment

In this section, we describe the components of our simulation model and its implementation aspects, as well validation and configuration issues. Our simulation model uses a discrete-event approach. We have validated our simulation model by comparing the experimental results of the simple queueing problems (M/M/1, M/M/2, and M/M/4) with their analytical solutions.

3.1 Components of the model

We define six basic components to build our simulation environment. Each component has some attributes to identify its state and several functions/methods to modify that state as well as to interact with the other components. Every component has an object association with one or more object instances. For example the generator component and the global scheduler have only one object instance. The components of simulator are listed below.

- *job*: The job object contains information about the attributes describing its characteristics.
- *resource* : The resource object contains information about machine attributes and several methods that represent the machine capabilities.
- *queueing* : The queueing object is responsible for managing the jobs in a queue. Each queueing system is represented by a single instance.
- *generator* : The generator object generates jobs of various types. There is only one generator object instance in the system.
- *g-sched* : The global scheduler object is responsible for scheduling the newly arriving jobs. There is one object instance in the system.
- *l-sched* : The local scheduler object schedules jobs received from the global scheduler. There is one l-sched object instance in each cluster.

3.2 Implementation

Our simulation environment was developed using the simulation package, C++SIM [2]. We use two additional random number generators to enhance the quality of randomness for some independent variables. The first generator is *the drand48() generator* (standard C library), and the second one is *mtGen generator* from Matsumoto [6]. *Drand48()* is used to generate the random variates from two-stage hyper-exponential distribution used for the execution time of the job. *MtGen* is used to generate the random variates from the exponential distribution used for the inter-arrival time of jobs. To generate the job types, we use the internal generator of C++SIM.

3.3 Validation

To validate our simulation model, we have performed experiments for some simple queueing problems. Here we show the results for an M/M/4 queueing system. The mean service time is four time units. The differences between the simulated results and the analytical theory are consistent with the r.m.s. errors in the simulations, derived from 16 independent runs for all simulated arrival rates. The measured standard deviations in the turn-around times (derived from the standard deviation within each run and the differences between the 16 runs) are also consistent with theory. Similar results were obtained for M/M2 and M/M/1 cases.

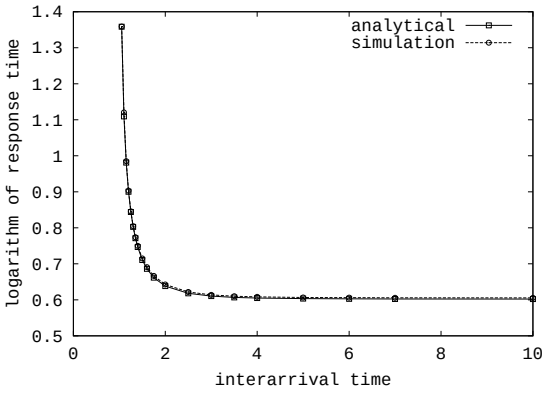

Figure 2: The performance of M/M/4 queueing case

3.4 Configuration

To use the simulation environment, we need to define the system configuration. For this purpose we have to model the resource, job structure, and the arrival processes. Furthermore, a set of performance criteria is needed. As the total number of parameters in the system already is quite large (global scheduling model, length of local queue, arrival rate, and local scheduling policy), we decided to keep the remainder of the model as simple as possible.

The resources consist of four identical clusters, and each cluster has four identical nodes (workstations). Every cluster is running a local scheduler and each node belonging to a cluster has a CPU scheduler. The system model is shown in Figure 1.

We distinguish two basic types of jobs, sequential jobs and parallel jobs. Sequential jobs consist of a

single task. Parallel jobs can have between two and four identical tasks, each is divided into sections taking one unit of CPU time. After each section, all tasks in the job synchronise (Figure 3).

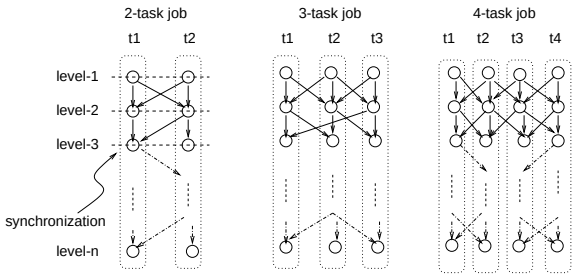

Figure 3: Communication model of parallel jobs

4 Simulation results

In this chapter, we show our simulation results of the selected measurements with a brief discussion. A more comprehensive discussion of all the results is given in the next chapter.

4.1 Parameters

Jobs are characterised by two main attributes: the number of tasks, and their minimum execution time (the CPU time required for the execution of a single task). The arrival process generates jobs of each type (sequential or two, three or four parallel tasks). For each type, a separate hyper-exponential distribution is used for the execution time. As experience shows that massively parallel jobs also tend to run longer [4], we have chosen the distributions accordingly.

Table 1: Distribution of generated jobs

# tasks	fraction	mean T	std dev.	work
1	0.7	4	5	2.8
2	0.1	4	5	0.8
3	0.1	8	10	2.4
4	0.1	16	18	6.4

Table 1 shows the parameters of the job generator. The amount of work in a job is the product of the number of tasks and the execution time (T). Other than for the frequently used exponential distribution of execution times, the standard deviation of the distribution must be specified for the hyper-exponential distribution. The specified distributions all have a longer tail than an exponential distribution. As the simulated system contains 16 nodes, 16 units of work can be processed per unit time. This corresponds to a minimum average job inter-arrival time of 0.775 time units. An important simplifying assumption is that the actual CPU time needed is available to the scheduler.

4.2 Measurement Strategy

We have chosen to consider the following parameters as the variables in our simulations :

- *Global queue model* : separate queues or a single global queue.
- *Global scheduler policy*: FCFS, SJF, or FF (when a single queue is used; SJF otherwise.)
- *Local queue size* : (zero, four, and eight jobs)
- *Local scheduler policy* : FCFS, SJF, or FF (when a local queue is used)
- *Job inter-arrival time* : from 0.8 to 1.5 time unit

We do not use all parameter combinations. By reseeding the random generators, we made 16 independent runs of 4000 jobs, with a start-up period of 200 jobs for each parameter combination. Except for their spacing in time, the same set of 16 times 4200 jobs was used for each parameter setting. System utilisation and run time were measured from the time the 201st job was submitted to the moment that the 4200nd job was finished. Performance statistics for each job were obtained at completion for the 201st job up to the 4200nd job. Job generation continued until the end of the simulation. As the scheduling system influences the order in which jobs are executed, there are differences in the population sampled.

For each run we have measured the average turn-around time for all types of (completed) jobs together and for each type of job separately, and the standard deviation of the turn-around times. We also measured the overall maximum turn-around time, the system utilisation, the number of jobs remaining in the global queue at the end, and the average of the slowdown factor (defined as the ratio between the response time of a job run on the shared resources and that on the dedicated resource).

4.3 Turn-around Times

The average turn-around time is a good indicator for the level of service to be expected from a system. We have measured the average turn-around times for each type of job (sequential, and two, three and four parallel tasks) and the average over all types. In Table 2 we present the measured values for the average turn-around times for all jobs for runs with mean job inter-arrival times of 0.8, 0.9, 1.1, 1.3 and 1.5 time units, corresponding to loads of about 97 %, 86 %, 71 %, 60 %, and 52 % , respectively. The table shows that, for the highest arrival rate (the inter-arrival time

Table 2: Reference model turn-around times T as a function of inter-arrival time. The errors given are the r.m.s. errors in the averages as derived from the spread in the computed averages for each of the 16 independent runs. They have not been computed from the internal values in each run, as those values may be highly correlated.

Global policy	Local policy	Local queue	0.8		0.9		1.1		1.3		1.5	
			T	error	T	error	T	error	T	error	T	error
S-SJF		0	24.3	0.5	18.2	0.6	11.6	0.2	9.1	0.2	7.9	0.1
S-SJF	FCFS	4	36.9	1.5	22.3	1.3	12.0	0.3	9.1	0.2	7.9	0.1
S-SJF	FCFS	8	46.7	2.3	25.0	1.7	12.1	0.3	9.1	0.2	7.9	0.1
S-SJF	SJF	4	26.5	0.8	18.8	0.7	11.7	0.2	9.1	0.2	7.9	0.1
S-SJF	SJF	8	26.2	0.7	18.7	0.7	11.6	0.2	9.1	0.2	7.9	0.1
S-SJF	FF	4	34.3	1.5	20.8	1.1	11.7	0.2	9.1	0.2	7.9	0.1
S-SJF	FF	8	41.2	2.5	22.1	1.4	11.8	0.2	9.1	0.2	7.9	0.1
L-SJF		0	53.7	5.6	22.2	1.6	11.6	0.2	9.1	0.2	7.9	0.1
L-SJF	FCFS	4	89.9	10.0	30.1	3.2	12.0	0.3	9.1	0.2	7.9	0.1
L-SJF	FCFS	8	90.2	10.5	29.1	2.9	12.1	0.3	9.1	0.2	7.9	0.1
L-SJF	SJF	4	31.6	2.2	18.9	0.8	11.7	0.2	9.1	0.2	7.9	0.1
L-SJF	SJF	8	26.2	0.7	18.7	0.7	11.6	0.2	9.1	0.2	7.9	0.1
L-SJF	FF	4	71.9	8.9	24.4	2.2	11.8	0.2	9.1	0.2	7.9	0.1
L-SJF	FF	8	60.7	6.6	22.7	1.8	11.8	0.2	9.1	0.2	7.9	0.1
G-FCFS		0	100.6	12.4	28.8	2.8	11.9	0.3	9.1	0.2	7.9	0.1
G-SJF		0	25.0	0.5	18.7	0.7	11.7	0.2	9.1	0.2	7.9	0.1
G-FF		0	87.5	11.4	25.1	2.3	11.8	0.3	9.1	0.2	7.9	0.1

of 0.8), the smallest queue first policy (S-SJF) with no local queue gives the shortest turn-around times.

In Figs. 4, 5 and 6, the turn-around times for all jobs, for sequential jobs and for four-task jobs are shown for the five scheduling policies without a local queue. As might be expected, the S-SJF policy does best for the sequential jobs as well (the large majority of the jobs is sequential), but L-SJF does much better for the four-task jobs.

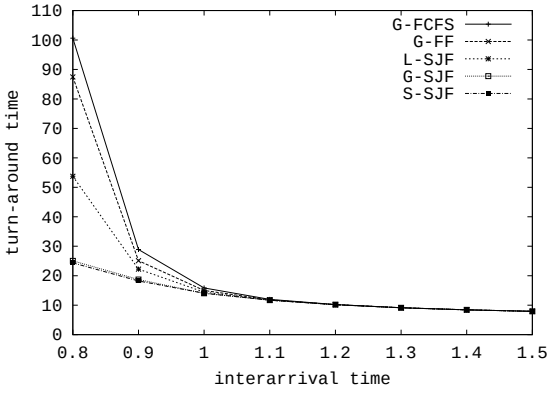

Figure 4: Turn-around times with five global strategies without local queue for all types of job

When a local queue is used (Figure 7), using an SJF policy for the local queue results in the shortest aover-all turn-around time, and FF appears to be the next best. Its bias against large parallel jobs results in longer turn-around times for those jobs.

4.4 Standard Deviation and Slowdown

The average turn-around times are not a good indicator for the ways the scheduling delays are distributed over the jobs. The standard deviation of the

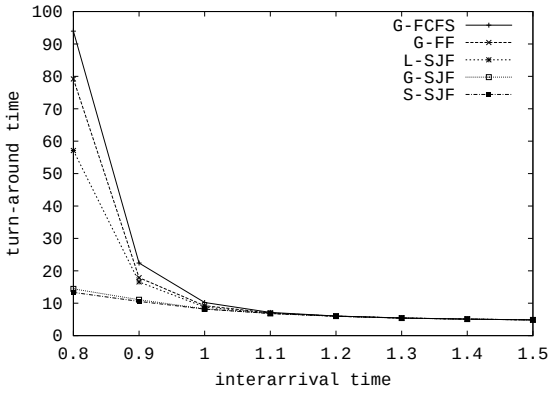

Figure 5: Turn-around times with five global strategies without local queue for sequential jobs

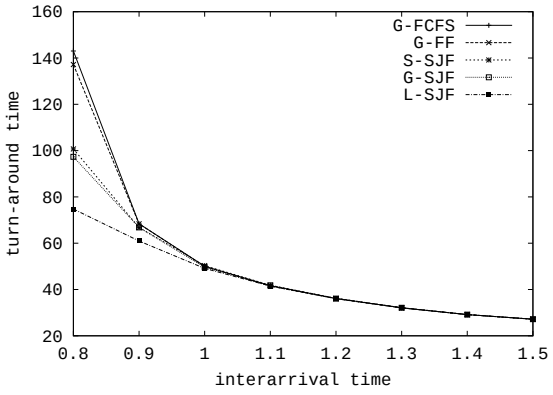

Figure 6: Turn-around times with five global strategies without local queue for four-task jobs

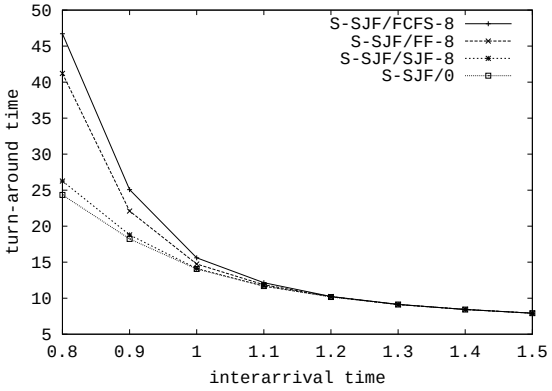

Figure 7: Turn-around times with S-SJF for various local policies

turn-around time is a measure that increases when jobs experience large differences in delays. This may indicate that the behaviour of the system is unstable, but it can also be due to a strong preference for one type of job above another. Slowdown is a measure for the delay experienced by a job, as compared to its minimal execution time (Figure 8). It tests the common expectation that small jobs should only experience little delay.

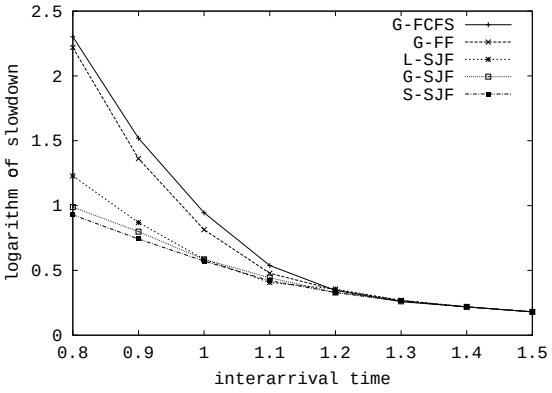

Figure 8: Slowdown of five global strategies without local queue. Note that the vertical axis is logarithmic.

The slowdown indicates that for the highest arrival rates, short jobs receive a much better service from all variants of the SJF policy. The spread in turn-around times also is somewhat smaller for these policies (data not shown).

4.5 Throughput

Our measurements of the system utilisation and jobs remaining in the queue (data not shown) indicate that for the highest arrival rate, the processing lags behind the arrival of work. This can be indicative of saturation. We have, therefore, performed additional experiments over 40 000 jobs for all models. For an inter-arrival time of 0.8 time units, the performance measures deteriorate in all cases, indicating that after 4000 jobs, the system has not yet reached its equilibrium state. Yet, average turn-around times, jobs remaining in queue and other indicators for saturation do not appear to grow out of bounds for any of the scheduling policies. At this arrival rate, the FF policy leads to the best system utilisation both at the global and the local levels, closely followed by FCFS. SJF continues to show short average turn around times and small slowdown values.

5 Discussion and Conclusions

In this paper, we have reported on simulation experiments on the effectiveness of various strategies for the global scheduler in our system. We have compared the use of a single queue for all types ob job with the use of separate queues. The purpose of splitting the global queue is to allow a more detailed control over the allocation of resources to jobs.

Averaged over all jobs, the results (Table 2) indicate that the turn-around time for the S-SJF policy (no local queue) is slightly shorter than that obtained for a single global queue with the same policy, G-SJF. Considering specific types of jobs, we find that for four-task jobs, the L-SJF policy is more profitable than S-SJF policy. On the other hand, the S-SJF policy gives the best performance for sequential jobs.

In general, the performance of S-SJF is better than L-SJF for high arrival rates (Figure 4). The SJF and FF policies show better performances than the FCFS policy almost for all jobs. If a good global policy (S-SJF) is used, the configuration without a local queue has a shorter turn-around than those using a local queue. Although most performance figures are better for the SJF policies than for FCFS, SJF can result in extreme delays for long-running jobs; S-SJF in particular is strongly biased against large jobs and can, in principle, result in starvation for those jobs. From the experiments, we conclude that the best strategy strongly depends on the desired performance characteristics.

In our paper, we have made a variety of simplifying assumptions that may influence the results.

- We have assumed no (variable) background loads for the clusters, and also that the expected work-loads for a job, as used by the scheduler, is a perfect predictor of the actual work-load. Dropping these assumptions should make delayed global scheduling (with a minimal local queue) even more profitable.

- We have assumed a homogeneous system. Dropping this assumption leads to a far more com-

plex scheduling problem, particularly if jobs with inhomogeneous resource requirements are allowed.

The results of these simulation experiments will be used in our research on run-time support systems, including our work on "Dynamite" [8], which supports dynamic load balancing for PVM and MPI jobs in clusters through the migration of tasks.

Acknowledgements

We gratefully acknowledge support for this work by the Dutch Royal Academy of Science (KNAW) under grant 95-BTN-15.
We wish to thank T. Basaruddin of the University of Indonesia and Benno Overeinder of the University of Amsterdam for fruitful discussions.

References

[1] D. L. Clark, J. Casas, W. Otto, M. Prouty, and J. Walpole. Scheduling of parallel jobs on dynamic, heterogeneous networks. http://www.cse.ogi.edu/DISC/projects/cpe, 1995.

[2] *C++SIM Simulation Package*, Univ. of Newcastle Upon Tyne, 1997. http://cxxsim.ncl.ac.uk/.

[3] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, and et. al. A resource management architecture for metacompting systems. In *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 62–82, Berlin, 1998. Springer.

[4] D. G. Feitelson and L. Rudolp. Metrics and bencmarking for parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 1–24, Berlin, 1998. Springer.

[5] A. Hori, H. Tezuka, Y. Ishikawa, N. Soda, H. Konaka, and M. Maeda. Implementation of gang scheduling on workstation cluster. In *Job Scheduling Strategies for Parallel Processing*, volume 1996 of *LNCS*, Berlin, 1996. Springer.

[6] M. Matsumoto and T. Nishimura. "mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator". In *ACM Transactions on Modeling and Computer Simulation*, volume 8. ACM, January, 1998.

[7] P. G. Sobalvarro, S. Pakin, W. E. Weihl, and A. A. Chien. Dynamic coscheduling on workstation clusters. In *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, Berlin, 1998. Springer.

[8] G. van Albada, J. Clinckemaillie, A. Emmen, J. Gehring, O. Heinz, F. van der Linden, B. Overeinder, A. Reinefeld, and P. Sloot. Dynamite - blasting obstacles to parallel cluster computing. In P. Sloot, M. Bubak, A. Hoekstra, and L. Hertzberger, editors, *High-Performance Computing and Networking (HPCN Europe '99), Amsterdam, The Netherlands*, number 1593 in Lecture Notes in Computer Science, pages 300–310, Berlin, April 1999. Springer-Verlag.

[9] A. van Halderen, B. Overeinder, P. Sloot, R. van Dantzig, D. Epema, and M. Livny. Hierarchical resource management in the polder metacomputing initiative. *Parallel Computing*, 24(12/13):1807–1825, November 1998.

[10] K. Y. Wang, D. C. Marinescu, and O. F. Carbunar. Dynamic scheduling of process groups. *Concurrency: Practice and Experience*, 10(4):265–283, April 1998.

[11] J. B. Weissman and A. S. Grimshaw. A federated model for scheduling in wide-area systems. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pages 542–550, Syracuse, NY, Aug. 1996.

[12] B. Zhou, R. Brent, D. Walsh, and K. Suzaki. Job scheduling strategies for networks of workstations. In *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *LNCS*, pages 143–157, Berlin, 1998. Springer.