Scheduling of Parallel Applications on Heterogeneous Workstation Clusters

B. Schnor S. Petri R. Oleyniczak H. Langendörfer
Department of Operating Systems and Computer Networks
TU Braunschweig, Bültenweg 74/75
38106 Braunschweig, Germany
schnor@ibr.cs.tu-bs.de

Published in Proceedings of PDCS'96, the ISCA 9th International Conference on Parallel and Distributed Computing Systems, pp. 330–337, September 25-27,1996, Dijon, France

Abstract

Scheduling and load balancing for parallel applications can be done at application level, or at system level. Application level scheduling is less transparent, because it has to be coded into each program, and can lead to contradicting decisions, when the different applications do not know about each other. Therefore, in this paper we focus on system level load balancing. We discuss the special properties of heterogeneous workstation clusters for scheduling parallel applications. A Shortest-Expected-Delay mapping is presented that assigns the processes of a parallel application to "virtually homogeneous" machines, based upon the current load situation. We present simulation results that show, how and when process migration is beneficial in heterogeneous systems.

Keywords: workstation clusters; heterogeneous systems; parallel applications; scheduling; mapping;

1 Introduction

A parallel application consists of a group of communicating processes which have to be assigned to the processors of a parallel system.

Algorithms for scheduling and processor allocation in multiprocessors for parallel processing are well studied. They can be classified into space–sharing and time–sharing disciplines [3]. Space–sharing disciplines are non–preemptive disciplines which partition the processors among the parallel jobs. Time–sharing disciplines use some form of round–robin–scheduling between the parallel jobs.

A workstation cluster may also be considered as a parallel computer. A number of research activities have tried to exploit the computing power of such environments [1, 4, 5, 7].

A popular tool to implement parallel applications on workstation clusters is PVM (*Parallel Virtual Machine*). With PVM, each user maps her application statically onto nodes which are specified in a configuration file. But in most cases, workstation clusters are not used exclusively by one user alone. This means that a number of applications compete for the resources. When scheduling is done on application level, this may lead to overload situations, when two applications decide independently to make use of the same host. Therefore, in this paper we focus on system level load balancing.

In the next section, we discuss the special properties of workstation clusters which have to be considered in the design of scheduling disciplines. In section 3, we present the *Shortest Expected Delay* mapping (SED) which regards the heterogeneity of the machines and also offers a simple computing model for the developer of parallel applications on heterogeneous systems. Finally, we introduce mapping state diagrams to illustrate the behavior of SED and present simulation results which show the benefits of SED.

2 The Challenges of Workstation Clusters

Scheduling on workstation clusters is quite different from scheduling on parallel computers. From the operating system view, each workstation is an autonomous system. Each machine has its own scheduler, most commonly a UNIX timesharing scheduler. The implementation of parallel job scheduling has to be in user

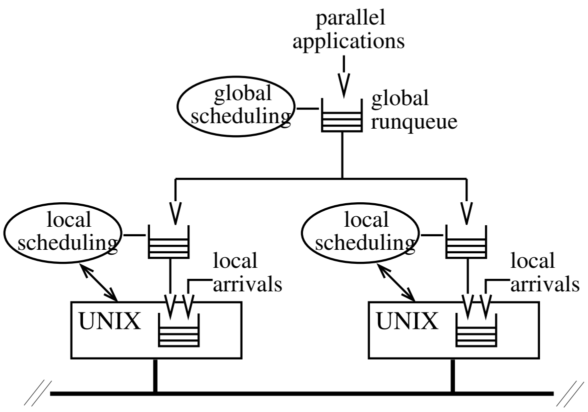

Figure 1: System Model.

space on top of the kernel. This leads to the 2-level system model shown in figure 1.

The first level is the UNIX timesharing system. Above this, there is a scheduling facility, which schedules only the parallel applications which are submitted to the system by a special user command. The global run queue for arriving parallel jobs is managed by a global scheduler, which is responsible for mapping the parallel jobs onto the machines.

2.1 Heterogeneity

The most challenging property of workstation clusters is their *heterogeneity*, since the speed characteristics of the machines may be different. If a parallel application is mapped onto a subset of the machines which consists of slower and faster machines, the slower machines may slow down the whole application, and the overall performance of the system goes down.

2.2 Motivation for Migration

While it has been shown that migration is not beneficial for load balancing in homogeneous systems [2], the situation is different in heterogeneous systems where applications can profit from migration to faster hosts.

Further, migration can improve fault tolerance, by evacuating hosts prior to regular shutdown, or through checkpointing.

We conclude that the scheduler has to get active in case of the following migration events:

- shutdown of a machine,
- interactive user arrivals,
- substantial load changes on a machine which would slow down the parallel job,
- substantially faster machines get available.

3 SED Scheduling in Heterogeneous Systems

The runtime of an application in a heterogeneous system depends on the speed capacity of the machines and their current load. First, we define delay factors to compare different machines. Then we introduce the concept of "virtually homogeneous" machines. Finally, the SED algorithm is defined.

3.1 Definition of Delay Factors

In a homogeneous system it is sufficient to have a reliable measure for the current load on the machines. In a heterogeneous system we wish to compare different architectures with different processing capacities. A load index suitable in a heterogeneous system is a *delay factor* which gives the expected delay of an application on a machine relative to, for example, the fastest machine in the system.

We assume for every machine M_i a given *speed factor* α_i, defined as follows. A sequential process which executes T time units on the fastest machine will need $\alpha_i \cdot T$ time units on machine M_i. We compare all machines relative to the fastest one. Hence, the speed factor of the fastest machine architecture is one.

If there are already n processes running on machine M_i and we assume a timesharing system with neglectable overhead, the estimated delay will be approximately $(1 + n) \cdot \alpha_i$. This motivates our definition of the *delay factor* of a machine M_i at time t as $d_i(t) := \alpha_i(1 + \mathrm{load}_i(t))$,
where $\mathrm{load}_i(t)$ is the number of runnable processes on machine i. For a discussion of this index see [6].

For example, a delay factor d_i equal to two means that the application will run two times slower on machine M_i compared to the fastest machine. If the machine M_i is idle, $d_i(t) = \alpha_i$.

3.2 Workload Model

We suppose a *strong synchronization model*, i.e. every process of a parallel job tends to synchronize with other processes of the parallel application, so that the slower machines will slow down the faster ones.

The speed-up of an application is limited by the highest delay factor d_{max} of all allocated machines. Therefore, the applications run on *virtually homogeneous* nodes with delay d_{max}. The delay of the application on n machines with delay factor d_j will be approximately $\frac{d_{max}}{n}$, if we assume an optimal parallelization.

For an example, we use a workstation cluster which consists of 5 faster machines ($\alpha = 1$) and 25 slower machines with $\alpha = 4$. We suppose that all machines are idle. If an application allocates only the 5 faster machines, the resulting delay is 0.2. If it allocates the 25 slower machines, it will be $\frac{4}{25} = 0.16$. Alternatively, the application can allocate all 30 machines. Because of the strong synchronization model, the application is mapped onto 30 "virtually homogeneous" nodes with $\alpha = 4$. The resulting delay is $\frac{4}{30} = 0.133$.

3.3 SED Scheduling Strategy

We present a mapping algorithm for heterogeneous systems which is an adaption of the *Shortest-Expected-Delay* mapping (SED) proposed for sequential processes. SED in homogeneous systems is known as "join the shortest queue". This is the optimal strategy in case of sequential processes in homogeneous systems [8].

The proposed SED mapping for parallel applications searches for "virtually homogeneous" nodes. Therefore, the heterogeneity of the system is transparent. This makes the development of a parallel application much easier.

The characteristics of SED are

- the global run queue is managed in FIFO order,
- processes of different parallel jobs may be mapped onto the same machine simultaneously, where they are scheduled by the UNIX timesharing system,
- migration, if faster machines get available,
- the user has to specify the maximum and minimum degree of parallelism (*minsize* and *maxsize*).

3.3.1 Definition of Delay Classes

All machines are classified according to their current delay factors.

We claim that the highest delay of a process is limited by the α_{max} value of the slowest architecture, i.e., SED maps at most one process onto a machine of the slowest architecture.

The possible delay classes depend on the speed factors of the machines. Each multiple of a speed factor which is less than or equal to α_{max} defines a delay class. Let α_k, $k = 1, ..., K$ be the different speed factors. The speed factors are ordered ($\alpha_1 = 1$, $\alpha_K = \alpha_{max}$). We introduce a δ-set, which consists of the possible delay classes:

$$\delta\text{-set} := \bigcup_{k=1}^{K} \{\alpha_k \cdot j \mid j \in \mathbf{N} \wedge \alpha_k \cdot j \leq \alpha_{max}\}$$

There are $maxclass := \mid \delta\text{-set} \mid$ delay classes. The order of the elements of the δ-set induces an order on the delay classes. The representative delay factor of delay class i is denoted by δ_i. In our example, we have $\alpha_1 = 1$, $\alpha_{max} = 4$ and $\delta\text{-set} = \{1, 2, 3, 4\}$.

When the load index of a machine changes significantly, the machine will also change its delay class.

3.3.2 The Availability Vector

The components a_i of the *availability vector* $(a_1, a_2, \ldots, a_{maxclass})$ give the number of currently available nodes in class i.

The *expected delay* D_k of a parallel job P_k which is assigned to machines M_j, $j = 1, 2, ..., n_k$, with current delay factors d_j is defined as

$$D_k := max\,\{d_j \mid j = 1, 2, ..., n_k\}\,. \qquad (1)$$

To determine the components of the availability vector, we have to consider the current delay factors $d_i(t)$ of the machines and their *slow–down threshold* $s_i(t)$ which is defined as follows. This threshold gives the maximum delay which can be tolerated on a machine M_i without slowing down one of the assigned applications. We define $s_i(t) := min\,\{D_k \mid \text{for all applications } P_k$ which have been assigned to $M_i\}$.

When the system is idle, $s_i(t) = \alpha_{max}$.

The availability vector is calculated as follows. The components are initialized with zero. If $d_i \leq s_i$, then $a_k := a_k + 1$ for all classes $k \geq d_i$. This is checked for every machine M_i.

The values of d_i, s_i and $(a_1, a_2, ..., a_{max})$ are updated whenever a parallel job is assigned or terminates.

In our example, when all machines are idle, the availability vector is $(5, 5, 5, 30)$.

3.3.3 The Mapping Algorithm

The SED mapping looks for the shortest expected delay under the current load situation, or in other terms it looks for the optimal speed up. SED maps an application onto the machines which are currently in delay class m, if

$$\frac{\delta_m}{a_{m_p}} = min\left\{ \frac{\delta_i}{a_{i_p}} \middle| \, i = 1, ..., maxclass \wedge a_{i_p} \geq minsize \right\},$$

where $a_{i_p} = min\,\{a_i, maxsize\}$.

If there is more than one delay class which fulfills this equation, we choose the one which reduces processor fragmentation, i.e., the fastest machine class which minimizes $a_i - a_{i_p}$.

3.3.4 Assignment of a Parallel Job

When a parallel job is assigned to some machines at time $t+1$, its expected delay D_k is determined. This may change the thresholds s_i of the allocated machines. Further, the delay factors d_i of these machines and the availability vector have to be recalculated.

In our example, the first arriving application P_1 may have $maxsize = 30$. Hence, P_1 will allocate all 30 machines with an expected delay of 4. The availability vector becomes $(0, 5, 5, 5)$. The next arriving job P_2 can still use the 5 faster machines without slowing down the first application.

3.3.5 Termination of a Parallel Job

When a job terminates, first the delay factors d_i of the concerning machines have to be recalculated. This may result in *upgrading* jobs which allocate some of these machines. Hence, the expected delays D_k of all jobs have to be updated.

In our example, when P_1 terminates, P_2 runs alone on the five fast machines, with a new expected delay of 1. Only the 25 slower machines stay available for the next arriving application.

Since the application is always running and contained
in $load_j$, we can not calculate D_k as in (1), but use $D_k = max\left\{\alpha_j(1 + \text{load}_j(t) - 1) | \, P_k \text{is ass. to } M_j\right\}$.

When a job is upgraded, the thresholds s_i of all machines have to be recalculated. This also effects the availability vector.

In our example, the availability vector is $(0, 0, 0, 25)$ after P_1 has terminated due to upgrading of P_2.

When the expected delays have been updated, the scheduler checks whether jobs are waiting in the run queue and may be mapped now. If the run queue is empty, the scheduler checks whether a running job can be migrated to machines of a faster delay class. This means that SED uses *migration* to speed up applications in case of a low loaded system.

4 Evaluation of Simulation Results

A performance measure which is still suited in a heterogeneous system is the mean residence time of the application, since it sums up the mean waiting time and execution time.

To get a better understanding of the behavior of the algorithms under different hardware characteristics, we use an artificial workload with constant service demands (100 min), constant $maxsize = 30$ and $minsize = 10\% \cdot maxsize = 3$. Since $maxsize$ is equal to the number of machines in the systems, the parallel jobs will act greedy and allocate as much nodes as possible.

4.1 Migration Costs

The approximation of the migration costs are motivated by the measurements with our experimental migration facility [4]. The time for synchronization and checkpointing before migrating a process is about 10 seconds. The checkpointing of several processes can be done in parallel on the different machines. After checkpointing, the processes are migrated to their new locations. The transfer time via TCP for a process of x KB can be approximated as $y = 0.4[s] + 0.0012[s/KB] * x$. This means about 12.7 seconds for the migration of a process of 10MB size.

The time which is needed for migration is the sum of checkpointing time and transfer time. If we migrate n processes of size 10 MB, the costs of migration are $10 + n \cdot 12.7$ seconds.

4.2 Mapping State Diagrams

We introduce mapping state diagrams to characterize the behavior of the SED algorithm. Mapping state diagrams are distinct from state diagrams which describe the system behavior in terms of number of jobs in the system.

A *mapping state* is a tuple $(m_1, m_2, ..., m_{maxclass})$ with one component m_i for each delay class. The component m_i gives the number of machines of delay class i which are allocated by some application.

The mapping state of the system changes when a job terminates or a job is assigned. A *mapping state diagram* is a directed graph where the nodes are the possible mapping states and the edges are the possible state transitions.

4.3 System 1

System 1 consists of 5 faster and 25 slower machines with $\alpha = 4$.

4.3.1 Mapping State Analysis

The mapping state diagram of system 1 under the given workload is shown in figure 2. As long as no job arrives, the system stays in the idle state $(0, 0, 0, 0)$.

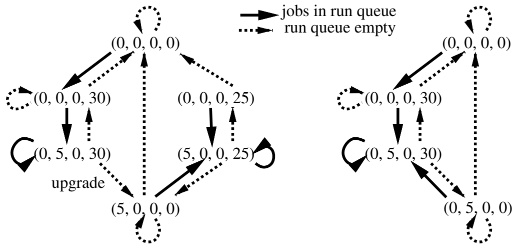

Figure 2: Mapping State Diagram of System 1: SED (left) and SED-NU (right).

When a job arrives, SED maps it onto all 30 machines with delay 4 and the system is in state $(0, 0, 0, 30)$. The availability vector is now $(0, 5, 5, 5)$.

When another job arrives, it will allocate 5 machines and the mapping state will be $(0, 5, 0, 30)$. When at least one of the two parallel jobs terminates and the run queue is empty, the system will leave this mapping state.

In case the job which allocates 30 machines terminates, there will occur an upgrading, since the remaining job runs all alone on the fastest machines. The system will change to state $(5, 0, 0, 0)$. The availability vector is $(0, 0, 0, 25)$. A new arriving job will allocate the 25 slower machines.

The mapping diagram shows that system 1 has the following characteristics under the given load situation.

- At most 2 jobs are running in parallel,
- jobs may run in timesharing mode (mapping state (0,5,0,30)),
- upgrading may occur, and
- there will be no migration at all.

We compare SED with SED-NU that uses no upgrading. The mapping state diagram of SED-NU is also shown in figure 2.

4.3.2 Simulation Results

The results of our simulations are shown in figure 3. The mean residence time, the mean waiting time, and the mean computing time of the jobs are given as a parameter of the job arrival rate. Further, the percentage of upgraded processes is shown.

For system 1, upgrading reduces both waiting and computing time. The percentage of upgrades decreases with increasing system load. This occurs, since upgrading is done only when the run queue is empty.

Nevertheless, upgrading leads to a better performance even under high load (see figure 6). The reason

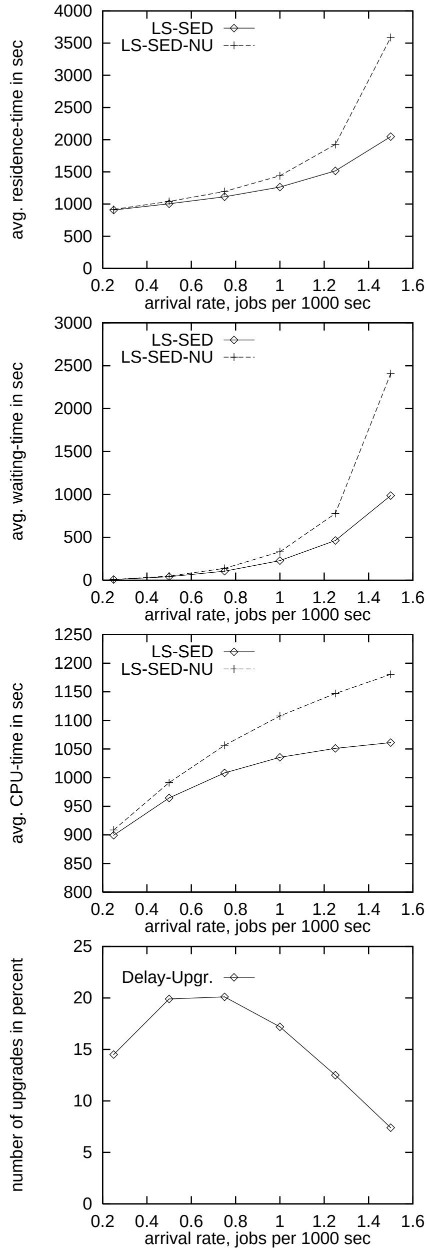

Figure 3: Simulation Results with System 1.

can be found by observing the corresponding mapping state diagram.

In case of SED-NU the corresponding mapping state for higher load is $(0, 5, 0, 30)$. The *asymptotic computing time* is

$$c_{limes} = \left(x \cdot \tfrac{2}{5} + y \cdot \tfrac{4}{30}\right) \cdot t,$$

where t is the mean service time demand of the jobs ($t = 100min$), and the weights x and y are the solution of the equations $x \cdot \frac{2}{5} = y \cdot \frac{4}{30}$ and $x + y = 1$. In this case there will run 3 jobs with delay $\frac{4}{30}$ while one job runs with delay $\frac{2}{5}$. This gives $c_{limes} = \frac{2}{10} \cdot t = 1200s$

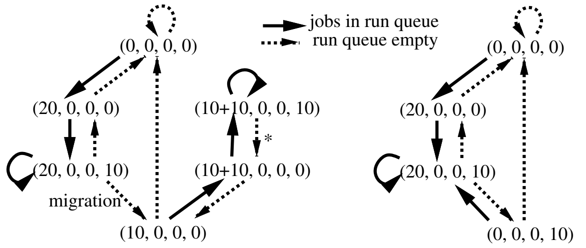

Figure 4: Mapping State Diagram of System 2: SED (left) and SED-NM (right).

(see fig. 3).

In case of SED the system may change between the states $(5,0,0,25)$ and $(0,5,0,30)$. The asymptotic computing time in state $(5,0,0,25)$ is $c_{limes} = \left(x \cdot \frac{1}{5} + y \cdot \frac{4}{25}\right) \cdot t$, with weights $x = \frac{4}{9}$ and $y = \frac{5}{9}$. The resulting asymptotic computing time is about 1060 s which is the upper bound of the SED curve in figure 3.

4.4 System 2

System 2 consists of 20 faster and 10 slower machine with $\alpha = 4$.

4.4.1 Mapping State Analysis

The mapping state diagram of system 2 is shown in figure 4.

Migration may occur when in state $(20,\ 0,0,10)$ the bigger application terminates and the run queue is empty.

Another transition where migration may occur is from state $(10+10,0,0,10)$ to $(10+10,0,0,0)$, when one of the applications on the 10 fast machines terminates and the run queue is empty.

The mapping diagram shows that system 2 has the following characteristics.

- At most 3 jobs are running in parallel,
- all machines are used exclusively, i.e., jobs never run in timesharing mode,
- there will be no upgrading at all, and
- migration may occur.

We compare SED with SED-NM that uses no migration.

4.4.2 Simulation Results

Our simulation shows the benefits of migration under low and medium load. Under low load most jobs will be mapped onto the 20 fast machines and migration occurs rarely. The percentage of migrated processes increases with increasing load. Under higher load the run queue will seldom be empty and the probability of migration decreases again.

Figure 5 shows that migration reduces the mean waiting time. SED also reduces the computing time up to an arrival rate of about 2.3 jobs per 1000 seconds. Here, the jobs benefit from the shorter runtime on the faster machines after migration. But for higher load, SED-NM leads to the smaller mean computing time. The reason is that the two strategies result in different asymptotic computing times.

Without migration the system will be most of the time in state $(20,0,0,10)$ under high load. The corresponding computing time is $\frac{1}{9}\left(8 \cdot \frac{1}{20} + 1 \cdot \frac{4}{10}\right) = 533,\bar{3}$ seconds. When SED is used the system may be in mapping state $(20,0,0,10)$ or $(10+10,0,0,10)$. The latter has a computing time of $\frac{1}{9}\left(4 \cdot \frac{1}{10} + 4 \cdot \frac{1}{10} + 1 \cdot \frac{4}{10}\right) = 800$ seconds which increases the mean computing time of SED compared with SED-NM.

The different behavior of upgrading and migration is summarized in figure 6 where the speed-up of SED against SED-NU resp. SED-NM is shown. The speed-up of SED against SED-NU is defined as

$$\frac{\text{res. time of SED-NU} - \text{res. time of SED}}{\text{residence time of SED}},$$

and similar for SED-NM. While the benefits of upgrading increase under higher load, SED with migration shows substantial benefits under low and medium load. It should be clear that the possible speed-up depends on the hardware characteristics of the system.

5 Conclusions

We presented a SED mapping which makes use of the heterogeneity of the system. SED is a non-preemptive discipline. The presented simulation results show the benefits of the upgrading and migration techniques used by SED.

Simultaneously, we are implementing a global Scheduler for PVM applications to verify our simulation results in a real environment. Here, the load of interactive users is also considered by the implemented SED algorithm. Further informations about our project are on our web side `http://www.cs.tu-bs.de/ibr/projects/load/` .

References

[1] C.H. Cap and V. Strumpen. Efficient parallel com-

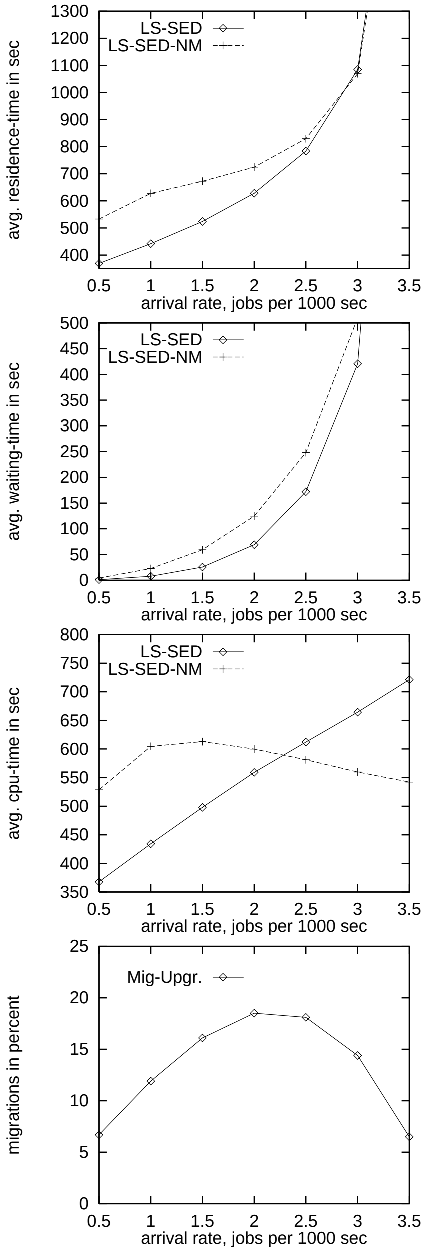

Figure 5: Simulation Results with System 2.

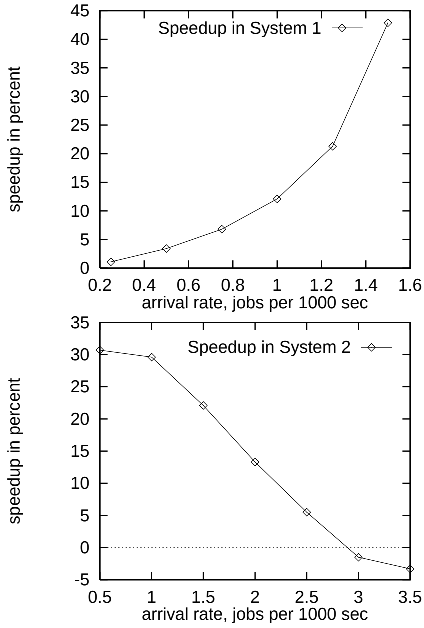

Figure 6: Speed-Up of SED.

puting in distributed workstation environments. *Parallel Computing*, 19(11):1221–1234, 1993.

[2] D. L. Eager, E. D. Lazowska, and J. Zahorjan. The limited performance benefits of migrating active processes for load sharing. *Performance Evaluation*, 6(1):63–72, 1988.

[3] D.G. Feitelson and L. Rudolph. Parallel job scheduling: Issues and approaches. In *Lecture Notes in Computer Science 949*, pages 1–18. Springer, 1995.

[4] S. Petri and H. Langendörfer. Load balancing and fault tolerance in workstation clusters – migrating groups of communicating processes. *Operating Systems Review*, 29(4):25–36, October 1995.

[5] J. Pruyne and M. Livny. Parallel Processing on Dynamic Resources with CARMI. In *Lecture Notes in Computer Science 949*, pages 259–278. Springer, 1995.

[6] B. Schnor, S. Petri, and H. Langendörfer. Load management for load balancing on heterogeneous platforms: A comparison of traditional and neural network based approaches. In *Proceedings of Euro-Par'96*, Lyon, 1996.

[7] V.S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, 1990.

[8] W. Winston. Optimality of the shortest line discipline. *Journal of Applied Probability*, 14:181–189, 1977.