Dynamic-SED for Load Balancing of Parallel Applications in Heterogeneous Systems

Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97), June 30 - July 2, 1997 Las Vegas, Nevada, USA

B. Schnor
Institute for Telematics
University of Lübeck
Ratzeburger Allee 160
23538 Lübeck, Germany
schnor@itm.mu-luebeck.de

M. Gehrke*
Ritterbrunnen 6
38100 Braunschweig
Germany

Abstract *In this paper, we present Dynamic-SED for load balancing in clusters of workstations. Dynamic-SED regards not only the current load of the machines, but also the currently free memory and the number of interactive users. It presents a new approach to achieve a co-existence between parallel applications and interactive users. We show that the migration rules of Dynamic-SED are circular-free. Results from a trace-driven simulation are promising, since the maximum number of migrations per hour is moderate.*

Keywords: dynamic load balancing, migration, workstation cluster

1 Introduction and Related Work

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 [6, 7, 9]. Workstations are user dedicated machines and only their idle times should be used for long running and parallel applications. The question is how both, parallel application and interactive user, can coexist.

We present and investigate a dynamic load balancing strategy called Dynamic-SED. It takes into consideration the current delay of the machines, currently free memory, and the number of interactive users.

Migration is the mechanism which is used to react on load changes. Migration strategies for load balancing are studied in [1, 5, 3, 10]. These studies investigate migration policies in homogeneous systems with no special care of parallel applications and interactive users.

In [4], a load balancing algorithm based upon the gradient model is proposed. The algorithm considers communication costs in its decisions, but no memory demands. Further, it does not take care of interactive users. Simulation results are presented for a 2-dimensional mesh and Hypercubes. The gradient model is not well suited for a workstation cluster, since the underlying network topology is a fully connected mesh.

In the next section, we present the Dynamic-SED discipline. In section 3, we investigate the stability of the migration rules. The behaviour of Dynamic-SED is tested in a trace-driven simulation. The results are given in section 4.

*Parts of this work were done within Marc Gehrkes master thesis [2] at the Institute of Operating Systems and Computer Networks at the Technical University of Braunschweig, Germany.

2 Dynamic-SED Discipline

The Shortest-Expected-Delay (SED) discipline was first presented in [8]. Here, we give an improved version of SED with a new definition of the availability vector. The advantage of using SED is that the heterogeneity of the system is transparent for the programmer of a parallel application.

Dynamic-SED is intended to fulfill the following two principles:

Principle 1: The user who wants to work at a workstation takes precedence over the parallel jobs.

Principle 2: The expected delay of the application shall be constant or decrease.

Our approach is the reservation of resources and migration to react on load changes.

2.1 The SED Mapping Algorithm

SED maps a parallel application onto *virtually homogeneous nodes*, i.e., onto machines with the same current delay. The *delay factor* of a machine M_i at time t is given as

$$d_i(t) := \alpha_i(1 + \text{load}_i(t)),$$

where $\text{load}_i(t)$ is the number of runnable processes on machine i, and α_i is a so-called speed factor which is a measure for the processor speed of machine i.

The machines are managed in *delay classes*. 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. A machine with current delay 1 counts as k "virtual nodes" with delay k. For all delay classes $i = 1, ..., maxclass$ and machines M_j, $j = 1, ..., n_k$,

$$a_{i_j} = max\left\{k \mid \alpha_j \cdot (k + \text{load}_j) \leq min\left\{\delta_i^{max}, s_j\right\}\right\}$$

is calculated, where δ_i^{max} is the upper bound of the delay class i and s_j is the *slow–down threshold* of machine M_j. This threshold gives the maximum delay which can be tolerated on a machine M_i without slowing down one of the assigned applications.

Table 1: Example of delay classes.

δ_i	δ_i^{min}	δ_i^{max}
1	1	1.5
2	1.5	2.5
3	2.5	3.5
4	3.5	4.5

The a_{i_j} are summed for all machines and give the component a_i of the availability vector. The availability vector is updated whenever a parallel job is assigned, terminates or migrates.

For example, a system consisting of 5 machines with $\alpha_1 = 1$ and 25 machines with $\alpha_4 = 4$ has the availability vector $(5, 10, 15, 45)$, when the system is idle. The delay classes with their representative δ_i, and upper and lower bound are given in table 1.

The user has to specify only the minimum and maximum degree of parallelism of her application (*minsize* resp. *maxsize*). SED maps an application onto the machines which are currently in delay class m, if $\frac{\delta_m}{a_{m_p}} = min\left\{\frac{\delta_i}{a_{i_p}}\middle| i \in \{1, ..., maxclass\} \wedge a_{i_p} \geq minsize\right\}$, where $a_{i_p} = min\left\{a_i, maxsize\right\}$. If there is more than one delay class which fulfills this equation, the fastest one is chosen.

We call $\frac{\delta_m}{a_{m_p}}$ the *expected delay time* (EDT) of the application.

2.2 Resource Reservation for Dynamic-SED

Dynamic-SED reserves some amount of resources for interactive users. This amount is chosen to be independent from the number of current users. Since interactive users tend to do other things like 'thinking', they will not run processes incessantly. Hence, the 'reserved resources' may be shared between the different users.

The parameters $load_{reserv}$ and mem_{reserv} give the percentage of cpu time and the amount of memory which shall be reserved for interactive processes. The parameter mem_{min} gives the lower bound for free memory. If the free memory drops under this bound, the probability of swapping is increased.

The reservation is done during the calculation of the availability vector. The components a_{i_j} for machine j with current load $load_j$ and current free memory $memory_j$ are calculated as follows. If there are interactive users, there is $\overline{load}_j = load_j + load_{reserv}$ and $mem_j = memory_j - mem_{reserv}$. Otherwise, there is $\overline{load}_j = load_j$ and $mem_j = memory_j$.

If $mem_j > mem_{min}$, we calculate

$$a_{i_j} = max\{k \mid \alpha_j \cdot (k + \overline{\text{load}}_j) \leq min\{\delta_i^{max}, s_j\}\},$$

otherwise, there is $a_{i_j} = 0$.

2.3 Migration Rules for Dynamic-SED

Migration strategies consist of a *selection* and a *location* policy. The selection policy determines the migration candidate, i.e., it determines which process on which machine has to be migrated. The location policy chooses the destination host for a migration candidate.

2.3.1 The Selection Policy:

The selection policy for Dynamic-SED is process-oriented. Since Dynamic-SED tries to guarantee the expected delay time EDT, the process becomes a migration candidate when the current delay gets significantly worse than the expected one. Further, a process may become a migration candidate if its host has less free memory capacity and swapping may occur. Finally, a high number of interactive users may cause a process to become a migration candidate.

Definition 1 *A process is called a* migration candidate *if at least one of the following properties holds:*

1. *The process was started with expected delay δ_i and the current delay of its host j is d_j with $d_j - \alpha_j > \delta_i^{max}$.*
2. *The process is running on host j with $memory_j < mem_{min}$.*
3. *The process is running on host j where the number of interactive users exceed the maximum number which are permissible on host j.*

Dynamic-SED checks, whether a process is a migration candidate over a period of time. This is done to avoid unnecessary migrations. Two different upper bounds are used to make the policy more sensitive in the presence of interactive users (choosing $c_{max}^1 > c_{max}^2$).

The *selection policy* works as follows:

1. Each process is periodically checked and its counter is updated: There is $c = c + 1$, if the process is a migration candidate, and otherwise $c = c - 1$.
2. If $c = c_{max}^1$ (no interactive users) resp. if $c = c_{max}^2$ (interactive users), the process is selected for migration.

2.3.2 The Location Policy:

The main location rule for Dynamic-SED is very simple.

> **Main Location Rule:** When the process was started in delay class i, then any machine which currently belongs to delay class i will be an adequate destination host.

When the delay class is empty, the process has to accept a slowdown. The set of possible destination hosts P is the set of hosts k without interactive users and enough memory, i.e., $memory_k > mem_{min}$. Otherwise, we would also migrate the problem.

The location policy depends on whether there are interactive users on the origin host or not.

Interactive Users on the Origin Host

In this case the process has to be migrated for the benefit of the interactive user.

Let j be the origin host. The destination host is the host with the lowest current delay which has at least as much memory as the origin host. Since p is already running on j, we have to recalculate the available memory. Let $memory\_usage(p)$ be the memory currently used by p, then $memory'_j := memory_j + memory\_usage(p)$, and

$$\begin{aligned} d_{dest} &= \min\{d_k \mid k \in P \wedge \\ memory'_j &\leq memory_k \wedge d_k < d^{max}\}. \end{aligned}$$

The upper bound d^{max} limits the number of processes on a host. Otherwise, single hosts without interactive users could be hopelessly overloaded.

If a destination host still cannot be found, the memory condition is dropped:

$$d_{dest} = \min\{d_k \mid k \in P \wedge d_k < d^{max}\}.$$

If still no destination host can be found, the user has to accept the situation.

No Interactive Users on the Origin Host

In this case, the process has dropped out of its delay class and/or memory is scarce. After migration the delay of the process should be substantially improved.

Now, the origin host j belongs to P. Since the process is already running on j, the delay of p on j is

$$d'_j := d_j - \alpha_j.$$

We claim that the delay of the destination host should be at least better than one α_j.

$$\begin{aligned} d_{dest} &= \min\{d_k \mid k \in P \wedge \\ memory'_j &\leq memory_k \wedge d'_j - d_k > \alpha_j\} \end{aligned} \tag{1}$$

3 Dynamic-SED is Ping-Pong–Free

Introducing migration rules into a resource management system there is the danger that processes are migrated around like nomads.

Definition 2 *We call a system of migration rules* migration-stable *if after migration of a process p from host A under current load situation $load_A$ to host B under current load situation $load_B$, will cause no migration with origin B.*

If a system of migration rules is migration-stable, no process on host B becomes a migration candidate, or, if a process on B may become a migration candidate, there will be no better destination host available.

Theorem 1 *The migration rules of Dynamic-SED are not migration-stable.*

Example: Consider the following situation. Let p be a process running on host A without interactive users. When p drops out of its delay class, Dynamic-SED migrates process p from A to host B. Hence, there are no interactive users on B.

Let q be a process on B with expected delay 1 and $memory\_usage(p) = memory\_usage(q) + \delta$, $\delta > 0$. Then q drops out of its delay class and gets a migration candidate. The destination host will be calculated due to equation 1:

$$\begin{aligned} d_{dest} &= \min\{d_k \mid k \in P \wedge \\ memory'_B &\leq memory_k \wedge d'_B - d_k > \alpha_B\}, \end{aligned}$$

where $memory'_B$ is the free memory on B without q. Since

$$\begin{aligned} memory'_B &= memory_B + memory\_usage(q) \\ &\quad - memory\_usage(p) \\ &= memory_B - \delta, \end{aligned}$$

the memory condition is less rigid and a host C which was not a suitable destination for the bigger process p may now become a suitable destination host for the smaller process q. Hence, q is migrated to C. $\diamondsuit$

A situation which should be avoided is one of circular migrations of processes.

Definition 3 *We say that a system of migration rules is* circular-free *if a migration of a* *process p from host A_0 to host A_1 will cause no migrations from A_i to host A_{i+1}, $i = 1, ..., n-1$ with $A_n = A_0$.*

Theorem 2 *The migration rules of Dynamic-SED are circular-free.*

Proof: Let us assume that Dynamic-SED is not circular-free. Then there exists a chain of hosts A_i, $i = 0, ..., n$ with $A_0 = A_n$ and A_{i+1} is destination of a process from A_i.

We will show the contradiction for three hosts A, B, C. The argumentation is analogous for longer chains. Let process p migrate from A to B, process q from B to C, and process r from C to A.

Since A is allowed to be a destination host, there are no interactive users on A. Hence, all destinations are determined due to rule 1.

Since p is migrated from A to B, there is

$$d'_A - d_B > \alpha_A > 0,$$

where d'_A is the delay on A without p.

C is destination for process q. Hence,

$$d''_B - d_C > \alpha_B > 0,$$

where d''_B is the delay on B without q. Since meanwhile p is running on B, there is $d''_B = d_B$ and we have

$$d_B - d_C > \alpha_B > 0.$$

From the migration of r it follows that

$$d'''_C - \tilde{d_A} > \alpha_C > 0,$$

where d'''_C is the delay on C without r but meanwhile with q running on C. Hence, $d'''_C = d_C$. The current delay $\tilde{d_A}$ of A is the delay of A without p which is equal to d'_A.

Therefore,

$$d_C - d'_A > \alpha_C > 0.$$

This gives a strong monotone decreasing chain of delays:

$$d'_A > d_B > d_C > d'_A,$$

which is a contradiction. $\diamondsuit$

It follows that processes will never be swapped between two hosts, i.e., a migration from A to B will never cause a migration of another process from B to A. We call this property *ping–pong–free.*

4 A Trace-Driven Simulation

To test Dynamic-SED, we did a trace-driven simulation. We were interested in the following questions:

- Can the delay of a parallel application be guaranteed within a tolerance interval during execution?
- How many migrations per hour do occur? This gives an approximation of the migration costs and whether the migration overhead is tolerable.

The trace data about the load situation, free memory and the number of interactive users were gathered on 28 Sun workstations. In the night experiment, the parallel application was started at 0:00 h and monitored over 4 days. In the day experiment, the parallel application was started at 4:00 pm and monitored over 4 days.

Both experiments were simulated for three parallel applications which differ in their $maxsize$: The first application A_{greedy} specifies $maxsize$ equal to the maximum number of available virtual nodes $a_{maxclass}$, the second application A_{70} will allocate 70 % of the available resources, and the third application A_{50} will allocate exactly 50 % of the available resources.

4.1 Model and Parameters of the Test System

The test model is based upon the following assumptions:

- The processes of a parallel application are computation-intensive.

In this case a parallel process will appear most of the time in the run queue. The current delay d_j of machine j is then calculated as

$$d_j = d_j^{real} + \alpha_j \cdot n_j,$$

where d_j^{real} is the monitored delay of machine j and n_j is the number of processes which are mapped onto this machine by the global scheduler during the simulation.

- The memory resources of processes which are mapped by the global scheduler during the simulation are 0.

  The reason for this assumption is the inaccurate load statistics of UNIX systems. The available memory statistic gives only a hint about the free memory and no correct informations. This is due to caching strategies which are in common use in current file systems. Pages which are belonging to the cache are subtracted from the free memory, while these pages are available.

- The action of the user in the presence of parallel jobs is 'ignore it'.

  The sampled trace data give the load situation in the test environment without parallel applications. Hence, the presence of interactive users is also monitored without parallel applications and the action of the user in the presence of parallel applications cannot be simulated.

  Therefore, we assume that the user keeps on working as long as he did due to the trace data.

Reservation Parameters:

$\boldsymbol{mem_{min}}$ **= 500 kBytes:** This bound was delivered from practical experiences with SunOS.

$\boldsymbol{mem_{reserv}}$ **= 1 MByte:** If mem_{reserv} is chosen to high, the machine may get unavailable while the interactive users may not make use of the resources. On the other hand, programs used by interactive users like `X`, `emacs`, or `netscape` tend to use several MBytes.

load_reserv = 0.5: Since interactive processes only need low computing power, *load_reserv* should be less than 1. Due to the local UNIX scheduling and their short runtime, they will be scheduled with a higher priority than the long running parallel application.

Migration Parameters:

$\boldsymbol{\delta T}$ **= 3 min:** The scheduler will check every three minutes, whether there is any migration candidate.

$\boldsymbol{c^2_{max}}$ **= 4:** When the process is a migration candidate over a time period of at least 9 minutes and an user is active, the process will be selected for migration.

$\boldsymbol{c^1_{max}}$ **= 6:** If no user is active, but the process is a migration candidate over a time period of 15 minutes, it will be selected for migration.

$\boldsymbol{d^{max}}$ **= 12:** This is the maximum number of processes which are permissible per host.

4.2 Migration Overhead

Figure 1 shows the number of migrations per hour over 4 days for the night experiment. The curves for the different applications are labeled by the corresponding job size. Migrations occur from about 10 h a.m. up to 7 h p.m. which corresponds to the working time of the staff and the students at the institute.

The observed maximum number of migrations per hour is 9 for A_{greedy}, 7 for A_{70}, and 5 for A_{50}. The mean number of migrations per hour over all 4 days is 1.27 for A_{greedy}, 0.61 for A_{70}, and 0.63 for A_{50}. This means that twice as many processes of the large application are migrated compared with the smaller applications. The number of migrations for A_{70} and A_{50} are almost the same.

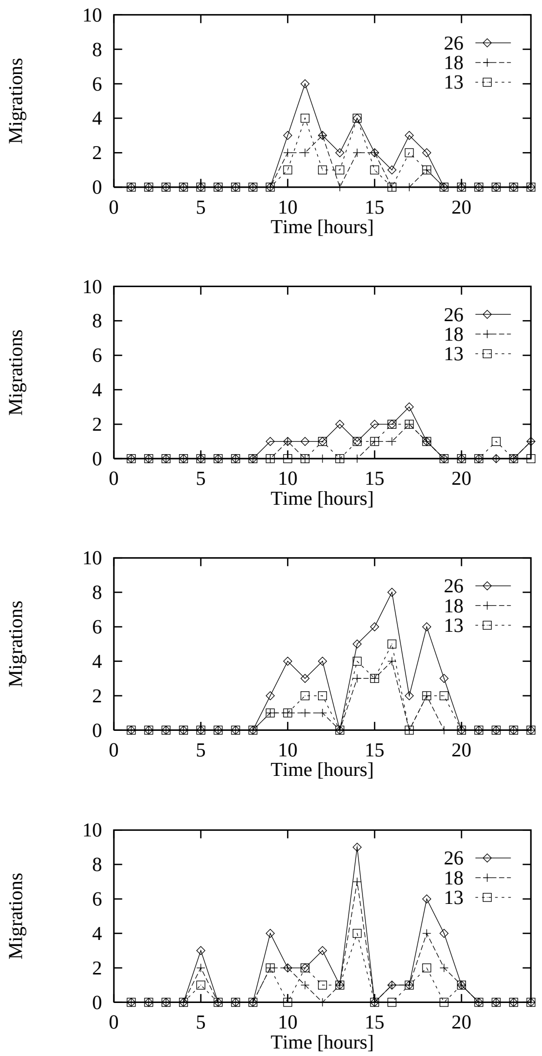

Figure 1: Number of Migrations over 4 days.

Table 2: Different delays of the applications for the night experiment.

appl.	size	MDC	MDT	EDT	sd
A_{greedy}	26	8.2	0.32	0.15	113
A_{70}	18	6.04	0.36	0.22	64
A_{50}	13	5.43	0.42	0.31	35

For the day experiment, the maximum number of migrations per hour is only 3 for A_{greedy} and 2 for the other ones. Since the applications are started when the system is already in its highest load situation, less resources are used and further, less migrations are necessary to guarantee a co-existence between parallel applications and interactive users.

4.3 Slowdown of the Applications

The evolution of the current delay of the application is summarized in table 2 and 3. For each application, the tables show the size of the application, the mean delay class (MDC), the mean delay time (MDT) which is the MDC divided by the size of the application, and the expected delay time (EDT). The slowdown (sd) of the applications is defined as

$$\mathrm{sd} = \frac{\mathrm{EDT} - \mathrm{MDT}}{\mathrm{EDT}}$$

and compares the expected and mean delay time.

For the night experiment, the mean delay time of A_{greedy} is 0.32 and the mean delay time of A_{70} is 0.36. This means that A_{70} is almost as fast as the greedy application. The slowdown of the greedy application is with 113 % the highest. This shows again that the application suffers significantly under high system load.

Table 3 shows the observed delays of the applications for the day experiment. The observed delays are closer to the expected delays. Since the sizes of the applications are much smaller, it was easier to hold the delay during the execution.

Table 3: Different delays of the applications for the day experiment.

appl.	size	MDC	MDT	EDT	sd
A_{greedy}	15	5.26	0.35	0.27	30
A_{70}	10	4.23	0.42	0.40	5
A_{50}	7	3.52	0.50	0.43	16

5 Conclusions

The experiments with Dynamic-SED have shown that migration is a useful tool for load balancing in a workstation cluster. We conclude that in the given test environment:

1. The migration rules add a tolerable overhead to the system, since the maximum number of migrations per hour is moderate.

2. The delay of greedy applications which allocate all in the idle system available nodes will increase significantly during execution.

3. When the application uses only about 70 % of the available nodes in the idle system, the runtime of the application will be approximately as good as the greedy application, but less migrations resp. migration wishes will occur.

4. When the applications are started at day time, the number of migrations are very low. This means that the reservation rules of Dynamic-SED have proven to guarantee a coexistence between parallel applications and interactive users.

References

[1] 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.

[2] Marc Gehrke. Ressourcemanagement für PVM–Anwendungen. Master's thesis, Institut für Betriebssysteme und Rechnerverbund, TU Braunschweig, 1996.

[3] Mor Harchol-Balter and Allen B. Downey. Exploiting lifetime distributions for dynamic load balancing. In *Proceedings of 15th ACM Symposium on Operating Systems Principles Poster Session (SIGOPS'95)*, page 236, December 1995.

[4] H.-U. Heiß. Dynamic Decentralized Load Balancing: The Particles Approach. In *Proceedings of the International Symposium on Computer and Information Sciences VIII*, Istanbul, 1993.

[5] P. Krueger and M. Livny. A comparison of preemptive and non-preemptive load distributing. In *Proceedings of the eighth International Conference on Distributed Computer Systems*, pages 123–130, 1988.

[6] 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.

[7] J. Pruyne and M. Livny. Parallel Processing on Dynamic Resources with CARMI. In *Lecture Notes in Computer Science 949*, pages 259–279. Springer, 1995.

[8] B. Schnor, S. Petri, R. Oleyniczak, and H. Langendörfer. Scheduling of Parallel Applications on Heterogeneous Workstation Clusters. In Koukou Yetongnon and Salim Hariri, editors, *Proceedings of the ISCA 9th International Conference on Parallel and Distributed Computing Systems*, volume 1, pages 330–337, Dijon, September 1996. ISCA.

[9] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concur-*

rency: Practice and Experience, 2(4):315–339, 1990.

[10] W. Zhu and P. Socko. Migration Impact on Load Balancing - An Experience on Amoeba. In *Proc. of the Fifth International Symposium on High Performance Distributed Computing*, pages 531–540, August 1996.