INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. ● Suite 600 ● Berkeley, California 94704-1198 ● (510) 643-9153 ● FAX (510) 643-7684

A Fuzzy Based Load Sharing Mechanism for Distributed Systems

Herwig Unger*, Thomas Böhme [†]

TR-98-026

August 1998

Abstract

This report presents a load sharing heuristic for distributed computing on workstation clusters. The approach is novel in that it combines the use of predicted resource requirements of processes (CPU-time, memory requirements, density of the I/O-stream) and a fuzzy logic controller which makes the placement decision. The heuristic is distributed, i.e. each node runs a copy of the prediction and load sharing code, and its implementation is based on PVM. Using a benchmark program (Choleski factorization) experiments were conducted to compare the proposed heuristic against standard PVM and an older version of the presented heuristic without the fuzzy logic controller.

*Department of Computer Science, University of Rostock, Rostock, D-18051 Rostock, Germany; Phone: +49 381 498 3403, Fax: +49 381 498 3366, E-mail: hunger@informatik.uni-rostock.de

[†]Department of Mathematics, Technical University Ilmenau, D-98684 Ilmenau, PF 10 0565, Germany, Phone: +49 3677 69 3630, Fax: +49 3677 69 3206, E-mail: tboehme@theoinf.tu-ilmenau.de

1 Introduction

Message passing systems such as PVM, Express, Linda and P4 offer the opportunity to use workstation clusters as an inexpensive alternative for distributed computing. However, to enhance the performance of such systems "additional system management software" is needed [11]. An important step toward this goal is to design algorithms for the distribution of processes among the nodes of a cluster. During the past decade a lot of work has been done in this field. (For an overview see [1], [9] and [11].)

The present contribution deals with a load sharing heuristic. In general terms, a load sharing algorithm works as follows: When a process p arrives for execution, the algorithm assigns p to a node of the cluster in such a way that the performance is good in some sense.

In order to enable such an action the load sharing algorithm needs to "know" the actual load of each node of the cluster. One solution is to poll all nodes of the cluster periodically. Since the load changes rapidly this has to be done frequently what causes much additional load and consequently, reduces the system's performance. Furthermore, it is not hard to see, that even frequent polling cannot guarantee the accuracy of a placement decision made without any knowledge about the system's future behaviour ([5], [8], [14]). A better solution is, therefore, to use a prediction. The proposed load sharing heuristic uses a simpler prediction scheme than the one used in [5] which is outlined in section 2.1.

The load of a node depends on several load factors such as the length of the CPU ready queue, the CPU-times of the processes in the CPU queue, the amount of memory in use, and the density of the I/O-stream. Since there is no suitable mathematical model that relates these factors with the performance of the system, most of the load sharing algorithms in the literature use a single figure to describe the load and distribute the processes in such a way that about the same load is assigned to each node of the cluster. The proposed heuristic uses a different approach. Three load factors are combined by a fuzzy logic controller which picks the node a new process is assigned to. A different approach which also involves a fuzzy logic controller is described in [6].

A load sharing system, called ALICE [1], using the proposed load sharing heuristic was implemented based on PVM. Here only those processes are subject to the placement decision that are created by a call of the `pvm_spawn`-function. An implementation on a command shell level and for MPI is in preparation.

An experiment has been conducted to compare the proposed load sharing heuristic against standard PVM 3.3.11 and an older version of the presented heuristic without the fuzzy logic controller. The results of the experiment are described in section 3.

2 Heuristic

The load sharing system ALICE consists of two parts - the predictor and the fuzzy logic controller (see fig.1).

[1]**A**daptive **L**oad Balancing System for **I**nhomogeneous **C**omputing **E**nvironments

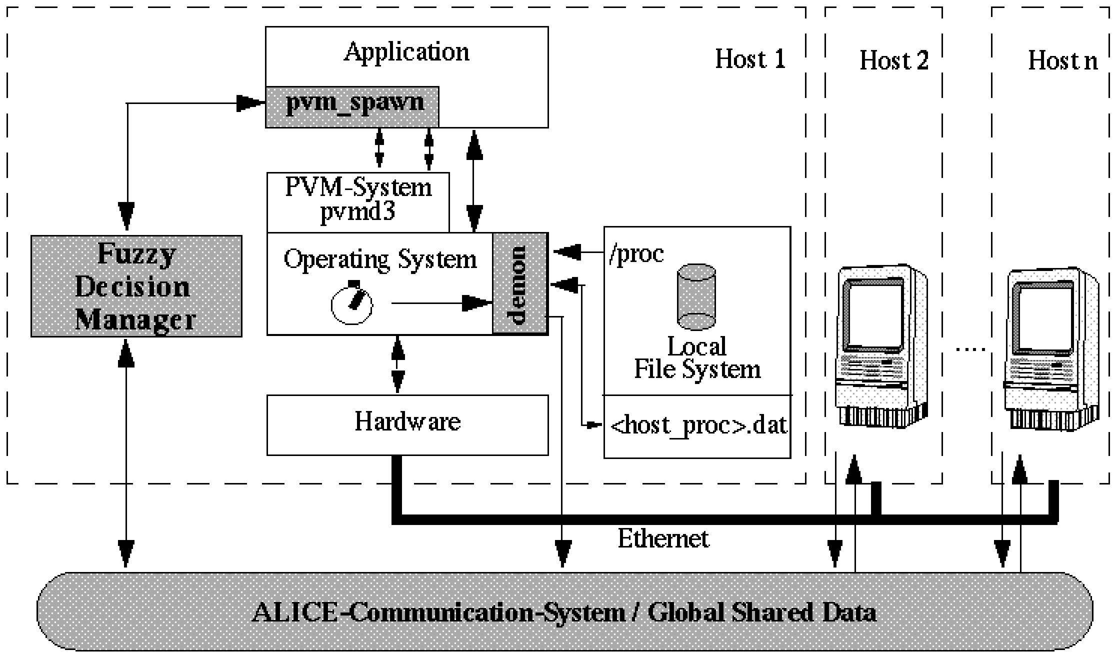


Figure 1: Structure of the Load Sharing System

The predictor is realized by a demon (called the local load prediction demon - llp-demon) which is incorporated in the operating system of each node. Since the decision manager must be invoked only if the `pvm_spawn`-function is called, the `pvm_spawn`-function has been substituted by a new function that includes the fuzzy logic controller.

In general the resource requirements of a process depend on the program the process executes as well as on the owner of the process. Furthermore, due to differences in the architectures of the processor nodes, one and the same process can have different resource requirements if it runs on different nodes. Therefore, a process p is identified by its program and its owner, and predictions of its resource requirements $\tau(p,i)$ (CPU-time), $\mu(p,i)$ (memory requirements) and $\omega(p,i)$ (density of the I/O-stream) are computed separately for each node i. Using these predictions of resource requirements of processes predictions of three load indexes of each node i are computed:

- $NIT(i)$ - a prediction of the time the processor is idle for the next time
- $MEM(i)$ - a prediction of the amount of memory in use
- $IO(i)$ - a prediction of the remainig I/O-capacity

The load sharing heuristics works as follow. When `pvm_spawn` is called on some processor node i, an identification of the program and the owner is sent to any other node. Every node j then returns its load indexes $NIT(j)$, $MEM(j)$ and $IO(j)$ as well as the predictions of the respective resources requirements $\tau(p,j)$, $\omega(p,j)$ and $\mu(p,j)$ of p.

These data are fed to the fuzzy decision manager which identifies the node p is executed on.

2.1 Prediction Scheme and llp-deamon

The llp-deamon consists of two parts (see fig.2.).

One part builds and maintains a list of the processes running on the respective node. In [9] it is shown that only processes that run longer than 1/3-1/2 sec generate a significant load and therefore, only processes running at least 1/2 sec are included in the list. The second part of the llp-deamon computes the predictions of the resource requirements of processes and the load indexes. It does not depend on the used platform what makes it easy to adapt the heuristic to other platforms.

The predictions of the resource requirements of a process p are weighted means of the actual resource requirements of the last N executions of this process on the respective node. In (1) the computation is shown for $\tau(p,i)$. $\omega(p,i)$ and $\mu(p,i)$ are computed in a similar way.

$$\tau_{new}(p,i) \quad = \quad e^{-\lambda}\tau_{old}(p,i) + \frac{t}{N} \tag{1}$$

The computation is done in a recursive way. $\tau_{new}(p,i)$ and $\tau_{old}(p,i)$ denote the new and the old value of $\tau(p,i)$, respectively, and t is the CPU-time of the last execution of p on the node i. The weight $e^{-\lambda}$ serves to limit the influence of the older values on the prediction, and is subject to an adaptation process.

The size of the memory in use, $MEM(i)$, of the processor node i is estimated by the sum of the memory requirements of those processes which are currently running on i. Using the predictions of the CPU-times of the processes running on the node i, $NIT(i)$ is computed subject to the following rules.

(i) At the start of the system $NIT(i)$ is zero.

(ii) When a new process p is started on i, $NIT(i)$ is updated by adding the prediction $\tau(p,i)$ of CPU-time of p.

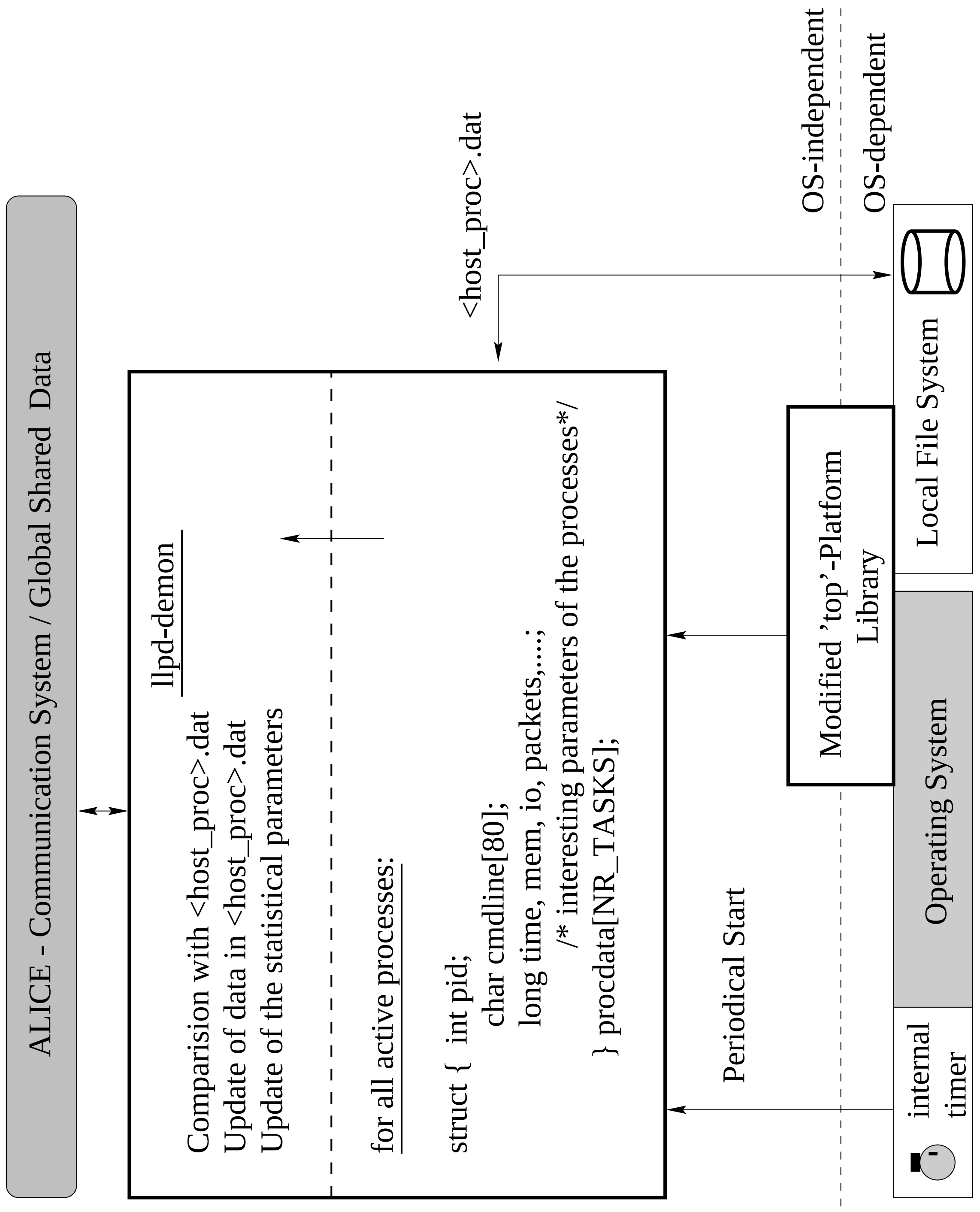
(iii) $NIT(i)$ is periodically decreased by the difference of the allocated CPU-time of the processes since the last update.

(iv) If a process terminates earlier or uses the CPU longer than predicted, $NIT(i)$ is corrected.

The prediction of the remaining capacity of the I/O-channel is computed similarly.

2.2 Fuzzy Logic Controller

The basic idea of the fuzzy logic controller is the following. In order to determine where to run a new process p, for each node i a real number $l(i)$ is computed in such a way that $l(i)$ is the greater the better the process p will fit on the node i. p is assigned to the node with the greatest $l(i)$. The main steps of the computation of $l(i)$ are outlined below.


ALICE - Communication System / Global Shared Data
llpd-demon
Comparision with <host_proc>.dat
Update of data in <host_proc>.dat
Update of the statistical parameters
for all active processes:
<host_proc>.dat
OS-independent
OS-dependent
Modified 'top'-Platform Library
Local File System
Operating System
internal timer
Periodical Start

```
struct {  int pid;
          char cmdline[80];
          long time, mem, io, packets,...;
               /* interesting parameters of the processes*/
} procdata[NR_TASKS];
```

Figure 2: Structure of the llp-deamon

1) Fuzzifications $NIT^f(i)$, $MEM^f(i)$, $IO^f(i)$ and $\tau^f(p,i)$, $\mu^f(p,i)$, $\omega^f(p,i)$ of the load indexes $NIT(i)$, $MEM(i)$, $IO(i)$ and the resource requirements $\tau(p,i)$, $\mu(p,i)$, $\omega(p,i)$, respectively, are computed.

2) Using a rule base $\mathcal{R}_1, ..., \mathcal{R}_s$, for each rule $\mathcal{R}_j$ a figure $l_j(i)$ is computed.

3) $l(i)$ is obtained from $l_1(i)$,...,$l_k(i)$ by a defuzzification algorithm.

It is well beyond the scope of this article to give an introduction to fuzzy logic (For more on fuzzy logic and fuzzy logic controllers the reader is referred to [16].) Nevertheless, some remarks should be made.

A fuzzy set in some set M is characterized by a function f from M to the closed interval $[0,1]$. f is called the membership function and, for an element x of M, $f(x)$ is considered to be the degree of membership of x in the fuzzy set. In order to define a fuzzification of a number y, finitely many fuzzy sets (called linguistic terms) are defined in the set y is taken from. If f_1,...,f_r are the membership functions of the linguistic terms, the fuzzification y^f of y (with respect to the linguistic terms f_1,...,f_r) is the vector $y^f = (f_1(y), ..., f_r(y))$. The linguistic terms are usually associated with linguistic labels such as *small*, *medium*, *large* etc.. Here five linguistic terms *very low*, *low*, *medium*, *high* and *very high* are defined for each input variable. The linguistic terms used for the output variable $l(i)$ are *very bad*, *bad*, *medium*, *good* and *excellent*.

The rule base consists of rules of the following type

$\mathcal{R}_k$: *If* $NIT(i)$ is very low *and* $MEM(i)$ is low *and* $IO(i)$ is medium *and* $\tau(p,i)$ is high *and* $\mu(p,i)$ is medium *and* $\omega(p,i)$ is low, *then* $l(i)$ is good

Given the fuzzifications of the input variables for each rule a figure is computed that describes the degree as to which the specific rule applies to the input. These figures define a fuzzy set which is then defuzzified using the center of gravity method. For details see [16].

The knowledge about the controlled system is encoded in the definition of the linguistic terms and the rules of the rule base. Thus the correctness of the decisions made by the fuzzy logic controller depends directly on the quality of the knowledge base formed by the linguistic trems and the rule base. In the present version of our system the knowledge base is based on experience.

3 Experiments

The above described load sharing heuristic was implemented on a cluster consisting of three PC's (i386 and i486) running under LINUX and connected via EtherNet. A distributed benchmark program performing a Cholesky factorization of a matrix taken from [13] was run with standard PVM, an older version of the presented load sharing heuristic ([14]) and the proposed heuristic. The CPU-time and memory requirements of the processes created by the execution of this program are strongly influenced by the calling parameters.

system	factor [sec]	fsolve [sec]	bsolve [sec]	total [sec]
PVM 3.3.11	10.0	14.1	12.0	30.4
old version	10.0	9.1	9.2	24.2
new version	9.8	8.8	8.9	23.8

Table 1: Additional processes with medium memory use and high CPU-load

system	factor [sec]	fsolve [sec]	bsolve [sec]	total [sec]
PVM 3.3.11	9.1	7.0	7.1	21.2
old version	9.1	7.3	7.4	19.9
new version	7.0	7.2	5.0	17.1

Table 2: Additional processes with high memory use and low or medium CPU-load

The nodes also run additional processes of different characteristics. The obtained results are shown in table 1 and table 2. It turned out that, especially if the additional processes have high memory requirements and low or medium CPU-time, the new load sharing heuristic achieves significantly better results than the older version without fuzzy logic.

4 Concluding Remarks

Using ideas from [4], it is planned to to apply a genetic algorithm to adapt the knowledge base to the specific requirements of the cluster in a subsequent version of the presented heuristic. Presently, the more sophisticated simulation of our system is in preparation. Furthermore, it is planned to apply the ideas exhibited in this paper to very large distributed systems.

References

[1] M. A. Baker, G. C. Fox, H. W. Yau, "A Review of Commercial and Research Cluster Management Software", Technical Report, Syracuse University New York, 1996, http://www.cs.rice.edu/ lwlutz/NHSEmain0.html.

[2] W. Becker, "Dynamische, adaptive Lastbalancierung für große heterogen konkurrierende Anwendungen", PhD-Thesis, University of Stuttgart, 1995.

[3] K. P. Bubendorfer, "Resource Based Policies for Load Distribution", PhD-Thesis, University of Wellington, 1996.

[4] O. Cordón, F. Herrera, et al., "Genetic algorithms ans fuzzy logic in control processes", Technical Report #DECSAI-95109, ETS de Ingeniería Informática, Universidad de Granada, March, 1995

[5] M. Devarakonda, R. K. Iyer, "Predictability of process resource usage: A measurement-based study of UNIX", *IEEE Trans. Software Eng.*, vol.15, no.12, Dec. 1989.

[6] S. Dierkes, L. Frese, "Load balancing with a fuzzy decision algorithm: Description of the approach and first simulations", Technical Report Universitt Dortmund, 1995.

[7] D. H. J. Epema, et al., "A world-wide flock of Condors: load sharing among workstation clusters", TR-95-130, TU Delft, 1995.

[8] K. G. Goswami, M. Devarakonda, R.K. Iyer, "Prediction-Based Dynamic Load Sharing Heuristics". *IEEE-Trans. Parallel and Distributed Systems*, Vol. 4(6), pp. 638-648, 1993.

[9] M. Harchol-Balter, A. B. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing", Technical Report TR-95-021, ICSI, Berkeley, 1995.

[10] G. Hipper, D. Tavangarian, "A new Architecture for Efficient Parallel Computing in Workstation Clusters", Int. Conference HPCN Challenges in Telecomp and Telecom: Parallel Simulation of Complex Systems and Large-Scale Applications, Delft, Netherlands, 1996.

[11] J. A. Kaplan, M. L. Nelson, "A Comparision of Queueing, Cluster and Distributed Computing Systems", NASA Technical Memorandum 109025, Langley Research Center, 1993.

[12] P. Mehra, "Automated Learning of Load Balancing Strategies for a Distributed Computer System", Technical Report, University of Illinois, 1993.

[13] B. K. Schmidt, V. S. Sunderam, "Empirical Analysis of Overheads in Cluster Environments", in: *Concurrency: Practice and Experience*, 1993.

[14] H. Unger, H. Unger, G. Hipper, "Stochastics and Learning Methods for Load Balancing in a Distributed Workstation Cluster System", in: Proc. of the Int. Conference of Applied Informatics, Innsbruck, 1997.

[15] G. Wilhelms, "Dynamische adaptive Lastverteilung für PVM mittels unscharfer Benutzerprofile", PhD-Thesis, Univ. Augsburg, 1994.

[16] H.-J. Zimmermann, *Fuzzy Set Theory and Its Application*, Kluwer Academic Publisher, Boston, Dordrecht, London, 1991.