EXPERIENCES SIMUALTING THE LOAD SHARING SYSTEM LYDIA WITH HIGH LEVEL PN

Herwig Unger

Universität Rostock, Institut für Technische Informatik
Albert-Einstein-Straße 21, 18051 Rostock, Germany
Tel. +49 381 498 3372, Fax +49 381 498 3366
E-mail: hunger@informatik.uni-rostock.de

Bernd Däne, Jürgen Nützel

TU Ilmenau, Lehrstuhl für Rechnerarchitektur I
I/A-TTI-RA, D-98684 Ilmenau, Germany
Tel.: +49 3677-692825; Fax: +49 3677-691614
email:{bdaene|nuetzel}@theoinf.tu-ilmenau.de

Keywords:

Distributed Systems, Load Sharing, Adaptive Systems, Petri Nets

ABSTRACT

A number of load sharing/balancing mechanisms was developed in order to improve the execution of a distributed application in a workstation cluster. The contribution presents an adaptive load sharing system basing on a load prediction for the computing nodes of the system. Using an online calculation of stochastic parameters of processes the quality of load sharing could be improved once more. The reachable results will be predicted using a Petri Net simulation. Furthermore the execution of a benchmark program in the new environment will be discussed.

1. INTRODUCTION

Distributed programming on workstation clusters became more and more important to solve a number of problems faster and more effectively. Obviously, the machines of such an architecture are not exclusively used by any user. Therefore a number of load sharing and load balancing mechanisms were developed to optimize the execution of processes on a workstation cluster architecture. (for an overview/classification see (Baker, Fox and Yau 1996), (Harchol Balter and Downey 1995) or (Kaplan and Nelson 1993)). These systems are implemented in such a way that they do not generate a too large processing and communication overhead for the system. Nevertheless, there remain a lot of other parameters influencing the efficiency, the quality and therefore the possible use of a load balancing or load sharing. A number of criterions for the evaluation of such systems are given in (Kaplan and Nelson 1993).
The strategy deciding on which machine new processes will be placed or to which one it will be migrated is very important for the characterization of the system. Mostly reactions of load balancing/sharing systems depend on existing load disproportions. Load balancings often have an additional problem to find out processes which are worth to migrate and to determine a suitable point for a necessary migration during their execution (Burchert et al. 1995) (Douglas and Ousterhout 1991).

In order to get as much as possible information about the programs for such decisions a number of systems like Condor (Epema et al. 1995) or LSF (Kaplan and Nelson 1993) sucessfully use so called *job submission files*. In these files the user can record typical resource requirements of a job like the average needs of memory, CPU-time or external devices. Although these information may increase the quality of a solution, a number of disadvantages can not be avoided, even if

- one program has different requirements to the system depending on its input parameters or time constants of extern processes connected with the program;
- users have typical working profiles which may be subject to change, especially while using a set of programs within a session (see also (Wilhelms 1994)) ;
- only general average load factors from the past or current load parameters are used to make a prediction if a machine has the opportunity to process a task in the future (LSF for instance places a new process on the machine with the lowest current load).

In the following contribution we intend to discuss a new method avoiding these problems and giving better solution for the placing of processes. Furthermore, before the implementation the advantage of the new approach shall be checked at first with a Petri-Net model.

2. BACKGROUND

It was mentioned that common load balancing/sharing methods do only react to load disproportions since the system simply considers the current state of the processing units. Furthermore, a large number of load profiles of the past are mostly wasted and thus not used for further computation (also description files will be mostly off-line calculated and therefore only a small number of older data will be used).
Nevertheless a lot of future load changes of a machine can be predicted. The reasons for a possible prediction are that .

- a number of processes are always started periodically from the system, the user or because there are a lot of repeating activities of the environment;
- special parts of the processes (e.g. iteration loops) cause a non-apriory-predictable but characteristic execution time which depends on the range of several parameters which are the input of the program from the current control application (such parameters are especially time constants for several activities of the technical process) or the current user;

- for a number of repetitions these parameters are constant or will not be changed quickly (for instance execution time of a C-compiler, since the program length is growing slowly).

Such relations of the processes change in time and can be registered online only with statistical means or off-line with so called work load generators. For realization, the following adaptive strategies are proposed to get a basis of the necessary performance prediction:

- a registration and learning of concrete load situations as described in (Mehra 1993), (Wilhelms 1994);
- using average CPU-time-profiles of the processes to predict and to use
  (a) the future load (Goswami, Devarakonda and Iyer 1993) without calculating the already done work of a process or
  (b) the time, when the machine will be idle again;
- a calculation of stochastic dependency parameters between processes.

Because of the simplicity we will follow the approach (2b). Furthermore we want to use - in contrast to some other systems (see Epema et al. 1995) and (Mehra 1993)) - online algorithms (as suggested in (Goswami, Devarakonda and Iyer 1993)) in order to enable a reaction to changes in the system during its work (e.g. changing execution times of programs/processes because of new values of calling parameters or input data from the environment).

The tasks of each load sharing/balancing system can be divided into two parts. At first all necessary system data has to be collected and in a second step they must be computed into an single cost parameter for each machine which allows to compare the abilities of the nodes and to make a suitable decision. Most of the process parameters can be easily collected from the system using the */proc*-file system or the *rwhod*-demon.
To get representive values of all parameters it is useful to calculate an average value. To avoid keeping much data this should be done in a recursive way. While doing so, one problem arises from the fact that the whole behaviour in the past will be kept and considered for calculating the average CPU-time. Therefore, this behaviour influences the precision of the current average time even if rapid changes occur. To improve this, the system must be able to forget older values from its history. This can be done if older values fall into oblivion by the use of an $e^{-\lambda k}$-function (In this expression λ is a constant value determining how fast the system will forget obsolete data). The formula below shows the examplary calculation of the average CPU-time $\tau_i(p)$ of a process on the *i*-th machine.

$$\tau_i(p) = \frac{\tau_{i-1}(p) \cdot N \cdot e^{-\lambda} + t_i}{N} = \tau_{i-1}(p) \cdot e^{-\lambda} + \frac{t_i(p)}{N} \quad (1)$$

Due to the fact that each event is estimated with a special oblivion factor, the number of considered events in the denominator has to be changed. Since each new event will be weighted with 1, the preceding event with $e^{-\lambda}$, the preceding event to that with $e^{-2\lambda}$ and so on, the number of considered values N in dependency of the oblivion is equal

$$N = \sum_{k=0}^{\infty} e^{-\lambda k} = \frac{1}{1 - e^{-\lambda}} \quad (2)$$

(when the system is in balance). Therefore to take N values in consideration the oblivion factor $e^{-\lambda}$ must be equal $(N-1)/N$.

The above considerations were made for the average CPU-time of processes. It is clear that all equations can be written in the same manner for other process parameters, e.g. the necessary memory space. Furthermore, in (Unger, Unger and Hipper 1997) a method processing dependencies between pairs of processes was demonstrated by the authors .

Now the current average CPU-time of each process can be used to determine the next time, when the respective machine will be idle (next idle time, NIT). The NIT is a value which is

- zero at the start of the system,
- updated by adding the average CPU-time of a process when it is started in the system,
- decreased periodically (by the use of the difference of the allocated CPU-time of the processes since the last update),
- corrected, if a process terminates earlier or uses the CPU about a longer range than predicted.

In such a way, the NIT is a prediction for the future load of each machine. Therefore, if the machine with the lowest NIT will get the next process an overall better run time behaviour in the cluster can be achieved. In section 3 we will show with a simulation that these coefficients can effort improvements while section 4 demonstrates that all functions can be programmed in an easy way without generating larger overheads and that the built system will be worked as expected.

3. Modeling and Simulation with High Level Petri Nets

3.1. Basics of the Petri Net Class Used in the Model

For evaluation and verification of the scheduling algorithm a formal model has been developed. The model uses a high level Petri net (Jensen 1992) class that is implemented in an object oriented manner in conjunction with the Tcl language (Ousterhout 1994).
Such ‚OPNTCL nets‘ (‚Object Petri Net Tcl‘) allow the use of program statements as arc attributes, either as guards for pre-arcs and as assignments for post-arcs. These statements are accepted in Tcl syntax and will be interpreted at simulation time. In this paper, for better readability we use a C-like syntax instead of Tcl. The operations work with data structures that are assigned to the individual tokens.
Furthermore, OPNTCL provides a module concept in such a manner that subnets may be connected with each other using shared places. In the graphical representation an arc without

arrows between places denotes these places to be ONE shared place. Modules can occur in multiple instances, with individual parameters if needed.
Arc labels are noted like ‚1 task x { x.time = 5 }‘, where ‚1‘ is the multiplicity, ‚task‘ is a predefined type of the token‘s data structure, ‚x‘ is a reference to the individual token, and the expression inside ‚{ }‘ constitutes a software statement. Terms like ‚x.time‘ refer to elements of the token‘s data structure. If a pre-arc is in consideration, the software statement constitutes a firing condition (‚guard‘).
A full explanation of OPNTCL including some additional features is available via World Wide Web (Nützel 1997).

3.2. Fundamental Structure of the Model

The model is developed in such a manner that it can easily be configured for various numbers of processing nodes. It cannot be explained in full detail in this paper. So this description exemplarily refers to a three-node-model and gives an overview plus a discussion of some selected details. The base of this model is the general structure of the system seen in figure 1.

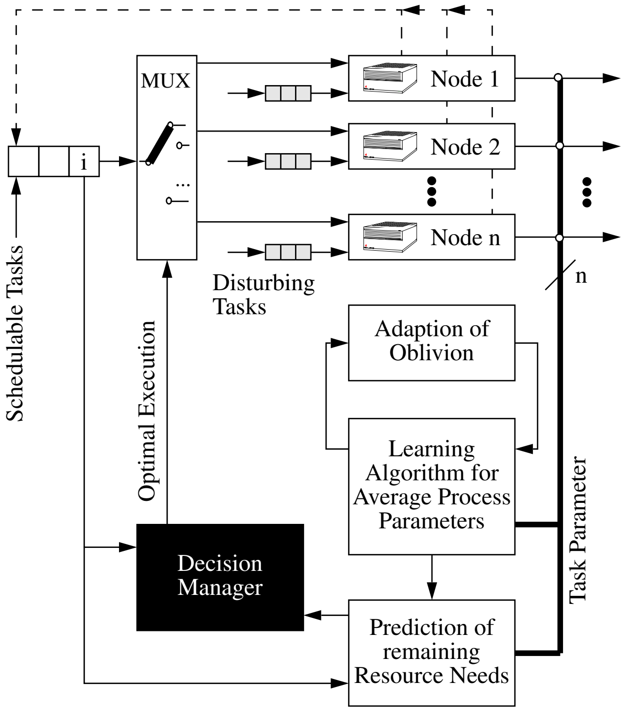

Figure 1: Structure of a Load Sharing with Online Learning

Tasks in this system can be executed on one of the n nodes. One class of (disturbing) tasks can not be scheduled, for instance because generated by the system/console user on the machine. Another class of tasks can be placed on the most suitable machine and will be executed there under a round robin scheduling. The execution parameters of all tasks will be collected and estimated on each node. On the base of these data the decision manager places new arriving, scheduleable tasks using the calculated NIT as described above.
Our model describes this system using the above decribed special class of high level Petri nets. Because qualitative and quantitative results of the execution of different tasks shall be considered, common queuing models cannot be used. The desired results have been derived by a simulative evaluation of the model.The fundamental structure of the model is shown in figure 2. It is based on a flow-oriented view where tasks are created, executed and consumed. There are two different types of tasks in consideration: ‚payload‘, the tasks to which the scheduling algorithm applies, and ‚other‘, (disturbing) tasks also running on the nodes independently of the scheduling algorithm.

Within figure 2 there is a number of modules (grey boxes). Places at the border of them are to be shared with others. A place shown in white color is ‚exported‘ by the module, e.g. it‘s properties are defined here. In contrast, ‚gray‘ places are ‚imported‘, e.g. they are references to another place.

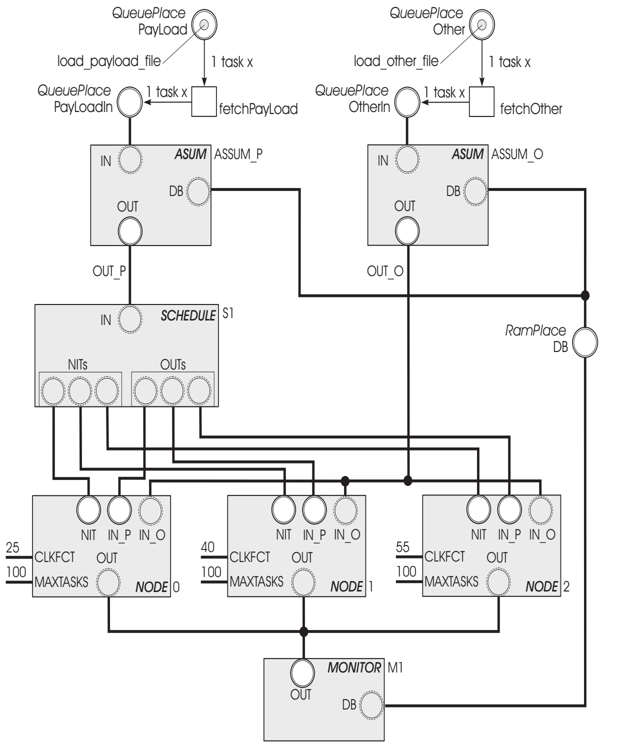

Figure 2: Module Level of the Simulation Petri-Net

The figure makes use of different place types provided by OPNTCL: ‚QueuePlace‘ is a place that handles tokens in FIFO manner. ‚RamPlace‘ allows random access to the tokens it holds, selectable by guards. If firing conditions are true for more than one token in a RamPlace the FIFO scheme applies (Nützel 1997). The task creation is controlled by pre-produced files that provide data about task type, start time and needed CPU-time for each instance of a task. For the ‚other‘ tasks information about the processing node which is to be used by the task is included too.

These files establish the simulation input data.
The file read is implemented in such a way that a token is created in the place ‚PayLoad‘ or ‚Other‘ every time a task has to start. The information about the individual task is written into the data structure of the token (type ‚task‘).
These tokens are transferred immediately to the modules ASSUM_P and ASSUM_O which make assumptions about the needed CPU-time of the tasks according to the scheduling algorithm. They share the place DB which holds the database about previous runs of the tasks. The algorithm does not read the ‚needed CPU-time‘ as mentioned above because this information in reality would be available only after the task‘s run. The ‚assumed CPU-time‘ is added to the token data.
The ‚payload‘ tasks enter the module S1 which completes the scheduling algorithm. For this reason the NIT‘s (‚next idle times‘) of all available processing nodes are compared and the token is transfered to the appropriate node. In contrast, the ‚other‘ tasks are statically scheduled according to their predefined node number (this is implemented by ONE shared place from which each node extracts ‚its‘ tasks).
Inside the three modules of type NODE the calculation of NIT and the execution of all active tasks in Round Robin cycles are modeled. These modules are discussed below in a more detailed manner. Parameters of these modules are the maximum number of tasks in the Round Robin cycle (MAXTASKS) and a speed parameter, expressed by the ratio ‚simulation ticks per node ticks‘ (clock factor, CLKFCT).
Tokens of completed tasks leave the nodes and enter the monitoring module M1, where the now-known ‚needed CPU-time‘ is used to update the database in DB. Furthermore, some observation data for each task (type, node, assumed CPU-time, needed CPU-time, total time and so on) is recorded for later evaluation.

3.3. Detailed Discussion of the NODE Module

To give an impression about the model‘s inside structure the module NODE now will be explained in more detail. NODE includes the model of the Round Robin cycle and the calculation and correction of NIT. For clarity a few statements are omitted that are needed for observation purposes only.

The module is shown in figure 3. To make visible which places are exported and which are imported by the module, these places are referred at the border of the module. Thick borders denote exported places and dashed borders denote imported places.
The place IN_O is the imported ‚source‘ of the ‚other‘ tasks. The guard of the following arc selects the tasks that belong to the actual node. In place IN_P the ‚payload‘ tasks are added by the S1 module which imports this place. IN_P represents the queue of tasks waiting for joining the Round Robin cycle.
When a task enters this cycle by firing the transition IN the node‘s next idle time is increased by modifying the token in the place NIT. This place holds exactly one token which‘s data structure (type ‚time‘) represents the calculated NIT for the node. The statement { y = y + x.asstime * CLKFCT } adds the assumed CPU-time of the actual task with respect to the actual node‘s CLKFCT to NIT.
The NIT value is decremented for each simulation tick by the transition tnit while it‘s greater then zero (the inhibitor arcs solve some unwanted conflicts). This value is available outside the module. .

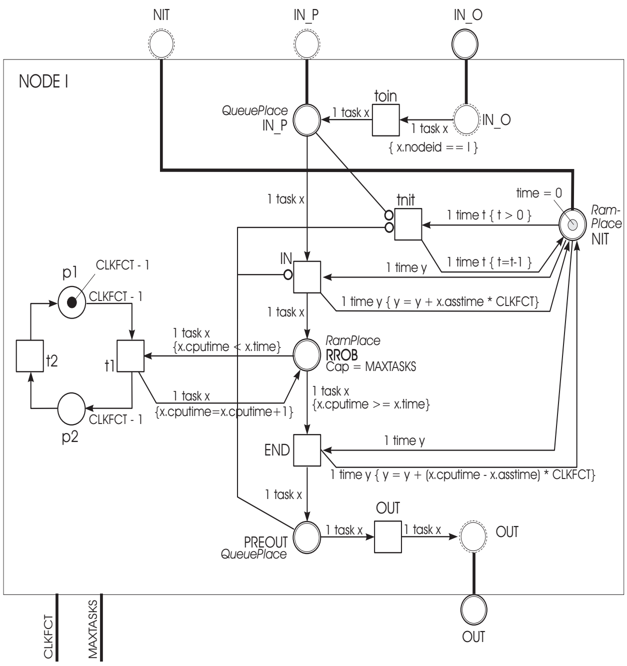

Figure 3: Round Robin Scheduling of Tasks on a Node

The place RROB holds all tasks that are in the Round Robin cycle. The elements p1, t1, p2 and t2 establish a count cycle which holds (CLKFCT-1) uniform tokens (‚black‘ tokens). Consequently transition t1 fires after each CLKFCT simulation ticks, e.g. it fires once for each node tick. When t1 fires, a task is executed for one time step. The real CPU-time of this task is incremented: { x.cputime = x.cputime + 1 }.

The cyclic activation of all tasks is managed by the FIFO behaviour of RROB. If a task‘s actual CPU-time reaches the predefined ‚needed CPU-time‘ the guard { x.cputime >= x.time } is activated and causes the task to leave the cycle by subsequent firing END and OUT. During this process NIT is corrected by the difference between ‚assumed CPU-time‘ and ‚actual CPU-time‘. The token is then dropped into the place OUT where it is accessed by subsequent modules.

3.4. Simulation and Results

The model has been simulated using input files that have been derived from former investigations. The files contained 525 ‚payload‘ tasks of four different types and 282 ‚other‘ tasks of five different types. The model included three processing nodes with different speed parameters (as in figure 2). The simulation has run in an Tcl environment under Windows NT and took about one hour on a Pentium-133 PC where the respective PN-simulation software was available. Furthermore, for comparision a second simulation without using the scheduling algorithm has been performed. In this case, the ‚payload‘ tasks were assigned to predefined nodes as the ‚other‘ tasks has been in all cases.

The below table 1 shows some results for the four types of ‚payload‘ tasks with and without scheduling. All time values use a time step of 0.5sec. The results in the below table clearly show the advantages of the scheduling algorithm.

Nevertheless it should be mentioned that the results are a really optimistic estimation, which cannot completely realized in a distributed system. The reasons are the simple set of selected tasks, the simple model of the task scheduler and last not least the fact that any communication or interaction and therefore waiting times of the tasks were neglected. The great advantage of the model is that it can be used in future to compare different load balancing strategies. The tested example has already shown that new strategies can be easily implemented while changing only a few components of the model and the problems of the easy model do not influence the relation between the achieved results too much.

Table 1: Results from the Petri Net Simulation

Absolute Task Parameters			Without Load Sharing	
Type of the Task	Total # of Starts	Σ Needed CPU-Time	Σ Total Execution Time	Avg. Ratio CPU/ Total
p1	78	1404	28386	0.049
p2	232	1160	23589	0.049
p3	116	3480	68932	0.050
p4	99	3136	74168	0.042
Absolute Task Parameters			**With Load Sharing**	
Type of the Task	Total # of Starts	Σ Needed CPU-Time	Σ Total Execution Time	Avg. Ratio CPU/ Total
p1	78	1404	10116	0.139
p2	232	1160	8638	0.134
p3	116	3480	24875	0.140
p4	99	3136	23253	0.135

4. Implementation and Experiences Using LYDIA-1

4.1. Structure of the System

Using the above positive results the Load Sharing System for Distributed Architectures LYDIA shown in figure 4 was implemented. It was designed for the use together with PVM-applications to improve their run time behaviour.

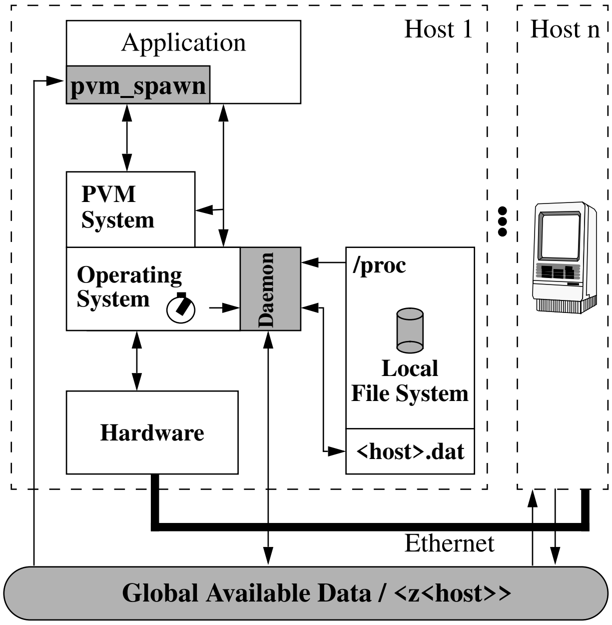

Figure 4: Structure of the Implemented Load Sharing System for Distributed Architectures (LYDIA)

The main part of this system is built by a local demon (*llpd*) included in the operating system of each machine. This demon periodically collects the data of all processes running at any time on the machine and calculates a statistics of average process parameters (including the average CPU-time as shown in the section above) in an only local available file. Using the results from (Harchol-Balter and Downey 1995), only processes that run longer than 1/3s-1/2s have to be considered to generate a significant load, therefore a longer sample time of 0.5 seconds will be used.
It is a task of our demon to calculate the above described NIT and to refine it periodically. The NIT- data from all machines of the current configuration is a global available data. This is realized by files in the global shared file system of the cluster (an extension using a data server or a broadcast will be implemented soon).

In such a way, the realized *llpd*-demon consists of two parts. The first one reading the information from the operating system has to be adapted to the operating system. Obviously it can use the */proc* file system or the *rwhod*-services to collect the necessary information. The other part calculating the current NIT in dependency of the new state of the system after each measurement and realizing the interface to the application via the PVM-system does not depend on the used platform.

This structure is one reason why applications written for several platforms can be used everywhere without any changes as described below.

The *pvm_spawn*-function of the PVM-Library is the only interface to the application. For doing so it was modified in such a way that it can read the NIT from all machines of the current configuration. Than these NIT-parameters will be compared for the placing of the new processes. It is clear that the machine with the lowest NIT will get any new process. In such a way a good load balance can be kept. Of course, starting a new process on a machine by *pvm_spawn* yields an update of the NIT of that machine.

Using the above described system derived and implemented on the base of the simulated Petri-Net model,

- written PVM-applications can be used without any modifications (only a new compilation is necessary);
- programs in a loaded cluster can be executed faster;
- a work in an inhomogeneous cluster is possible and the different performance of each machine will be considered for load sharing decisions;
- a group of possible target machines can be selected for the processes;
- changes of process characteristics will be learned by the system and improve future load sharing between the processors since a more precise load prediction will be realized.

4.2. Measurement Results

To get an impression of the advantage and the performance of the realized load sharing approach it was used with some benchmark programs which are known from (Schmidt and Sunderam 1993) or (Hipper and Tavangarian 1996) on a small distributed system (2 machines with LINUX). From the number of examples the Cholesky Factorization was been elected, because the run time of this program can be strongly influenced by the calling parameters (matrix- and block size). Because the system would be able to adapt changing parameters of the program, only after a first learning phase the execution time was measured. Therefore fixed values calling parameters of the program were used in different system situations. The positive results using the suggested new load sharing approach in a loaded cluster are to be seen in table 2 .

Considering the computing overhead of the demon a larger additional load could not be found. Since very short processes will be neglected, about 30-50 processes will be included in the local process table (file: <host_proc>.dat).
While using the last available statistics of each process the exactness of the NIT will be much higher than the use of any static value. From the experience with several applications the reachable average exactness of the NIT is about 5%-8%..

Table 2: Average Measurement Results for the Cholesky Factorization (Schmidt and Suderam 1993)
10 Processes, Matrix Size 200, Block Size 10

Type	factor [s]	fsolve [s]	bsolve [s]	total [s]
1.	7.0	6.9	6.0	18.0
2.	7.1	7.0	6.0	18.1
3.	10.0	14.1	12.1	30.8
4.	10.2	8.7	9.0	23.6

Types:
1. Standard PVM 3.3.11, unloaded system
2. PVM with Load Sharing, unloaded system
3. Standard PVM 3.3.11, disturbing processes on one node
4. PVM with Load Sharing,disturbing processes on one node

5. Conclusion and Outlook

The load sharing system LYDIA was implemented for the use together with the Parallel Virtual Machine (PVM). The succesful work of the new approach could be demonstrated as well by using a Petri net simulation as by evaluating the execution of a known Benchmark program. Furthermore, besides the good simulation results the advantages of Petri nets to describe difficult distributed system architectures could be proved once more.

While working with the described software three directions of extensions/further works for the future were derived. The first one concerns a reimplementation for much more platforms/versions of the operating system. A second step will be to allow a load sharing of batch jobs by including the mechanisms into a command shell environment. Last not least we intend simultaneously to consider and to improve the quality of the description of the required resources of the system in a third step.

REFERENCES

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

F. Burchert; M. Koch; G. Hipper; D. Tavangarian. 1995. „Task Distribution in a Workstation Cluster with a Concurrent Network". *Proceedings of APPT'95*, Beijing.

F. Douglas; J. Ousterhout. 1991. „Transparent Process Migration: Design alternatives and the sprite implementation". *Software-Practice and Experience,* 21(8).

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

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

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

G. Hipper, D. Tavangarian. 1996. „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)

K. Jensen. 1992. „Coloured Petri Nets". Springer Verlag, Berlin, New York.

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

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

J. Ousterhout. 1994. „Tcl and the Tk Toolkit". Addison-Wesley

B.K. Schmidt; V.S. Sunderam. 1993. „Empirical Analysis of Overheads in Cluster Environments". *Concurrency: Practice and Experience*.

H. Unger, H.V. Unger, G. Hipper. 1997. „Stochastics and Learning Methods for Load Balancing in a Distributed Workstation Cluster System". *Int. Conference of Applied Informatics*, Innsbruck.

G. Wilhelms. 1994. „Dynamische adaptive Lastverteilung für PVM mittels unscharfer Benutzerprofile". PhD-Thesis, Univ. Augsburg, Germany

J. Nützel. 1997. „OPNTCL-Object Petri Net Based TCL". http://www.theoinf.tu-ilmenau.de/opntcl/

BIOGRAPHIES

Herwig Unger was born in 1966 in Halle/S. He got its MSc graduation from the Technical University of Ilmenau in 1991. In 1994 he received the PhD from the TU Ilmenau with a work about transformation and implementation methods of Petri-Nets used as models of distributed programs. After a tenure-track position at the rank of an Assistant Professor at CS-Deptartment of the University of Rostock he got a permanent postion there in 1996. His reserach interests include resource scheduling, operating systems and formal methods for distributed computation and simulation.

Bernd Däne was born in 1962 in Jena, Germany (former GDR). He studied at the Technical University of Ilmenau (Germany) and received the master degree in Electrical Engineering in 1986. In the year 1990 he received the PhD with a thesis about memory architecture. Since this time he works as a scientist at TU Ilmenau, doing some work in Computer Science research and education. Special fields are: Embedded systems, design and modeling of various systems with Petri nets, distributedarchitectures.

Jürgen Nützel was born in 1967 in Schweinfurt, Germany. He studied there Electrical Engineering (bachelor) and Automation Engineering at the Technical University of Ilmenau (master). After an industrial period of 3 years as hardware designer at the Automation Group of Siemens he started in 1994 his scientific work at the TU Ilmenau. In 1998 he will finish his PhD. The topic is about the object-oriented design of distributed embedded real-time systems.Further special interest fields are: Petri nets and their simulation, parallel programming, Delphi, Tcl/Tk, CAN-Bus.