An Adaptive Load Balancing for Performance Improvements in Distributed Architectures

Herwig Unger[†] and Bernd Däne[‡]

[†] University of Rostock, Fachbereich Informatik, 18051 Rostock, Germany; hunger@informatik.uni-rostock.de, Fax: +49 381-4983440
[‡] Technical University of Ilmenau, I/A-TTI-RA, 98684 Ilmenau, Germany; bdaene@theoinf.tu-ilmenau.de, Fax: +49 3677-691614

Abstract

In many cases, the execution of a parallel program will be strongly influenced by data dependencies and processes running at the same time on the machine or network. These factors cannot be predicted. Therefore, powerful load balancing procedures are necessary in order to improve the run time behaviour. The disadvantage of this more efficient dynamic approach arises from a large communication or management overhead and delayed activities. In our contribution we intend to present an improved dynamic, decentral method for hierarchical automation appliances using idle times of the processor and the communication system for the realization of an efficient load balancing.

1 Introduction

Technical and technological processes are usually connected with a strong exchange and processing of large amounts of data in real time. Software for such automation appliances is often characterized by the following topics:

- a lot of sequential activities can be found which can be executed in a parallel manner
- these activities are depending on several conditions and data exchanges
- a hierarchical structure where waiting times of procedures can be found
- the execution requires only sometimes a special processing unit preferably if the process reads or writes data to or from I/O.

So a lot of software solutions can be more efficiently solved by parallel or distributed computing architectures. While doing so, it happens very often that the load caused by the application and related activities is concentrated on a special node in accordance with the flow of information or materials in a technical or technological process. The control flow will be significantly influenced by:

- the actual working capacity of the technical process or a number of its stages (e.g. in an assembly line)
- the availability of data from I/O devices and
- run time data dependencies (e.g. duration of iteration loops, number of generated clients, alternating sequences of instructions).

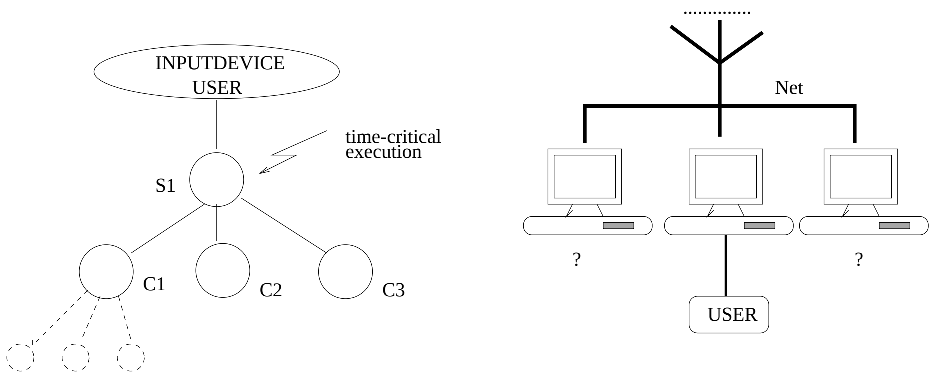

Figure 1: A devide and conquer system and its hardware.

In order to take these facts in consideration, load balancing mechanisms were developed. These methods are able to adapt the execution of the program to the actual state of the computer system. In the following part, advantages and disadvantages of existing load balancing methods will be discussed with respect to the needs of the applications, most of them have the structure of our example shown in figure 1. A number of clients will be managed by a server process. These clients may have subclients, too. It is clear that the execution of the server process influences the clients, especially if it cannot process the incoming amount of data in a short time. So it is useful to have powerful processing units for solving this task which should not have a large load caused by other processes.

2 Related Works

There are a lot of approaches for load balancing or load sharing to be found in the literature (see [1] for an overview). All available methods fall into two basic types: preemptive and non-preemptive ones. Since the non-preemptive ones (e.g. [9]) can be considered as an easy placement of new processes not allowing a process migration after start, it is obvious that these methods cannot be used for our work. Preemptive systems allow to suspend running processes and to move them to another remote host (e.g. [10]). Some of these systems use a-priori information about the processes, other ones only run time data such as the life- and CPU-time of a process. Although the execution of a number of parallel programs can be improved by the use of these approaches, there are a lot of open questions and problems. In most cases there are:

- a high communication or management overhead combined with a high latency low bandwidth LAN (see [6]),

- a very late reaction on an already existing load disproportion
- an activity of the balancing at a wrong point (e.g. a useless migration of a process which has almost finished)
- a lot of problems in order to collect information from the operating system without the addition of load and to determine, which processes are worth to migrate ([3]) and
- a high expenditure for process migration (because of additional network and CPU load).

From the authors' point of view, these problems result as well from the fixed structure of sequences of instructions of the processes and the communication in common parallel programs as the exclusive use of either a-priori or runtime information.

Therefore a new load balancing approach can be derived using a more flexible interpretation of the elements of a common parallel program influenced by the run time state of the used computer system. For implementations we use the well known C-extension PVM[1] (see also [8]) which was successfully applied to a lot of applications.

3 A Load Adaptive Kind of Programming

3.1 Foundations

Lets consider a special situation which can be graphically represented in a time diagram as well as in a Petri net ([2]) in figure 2.

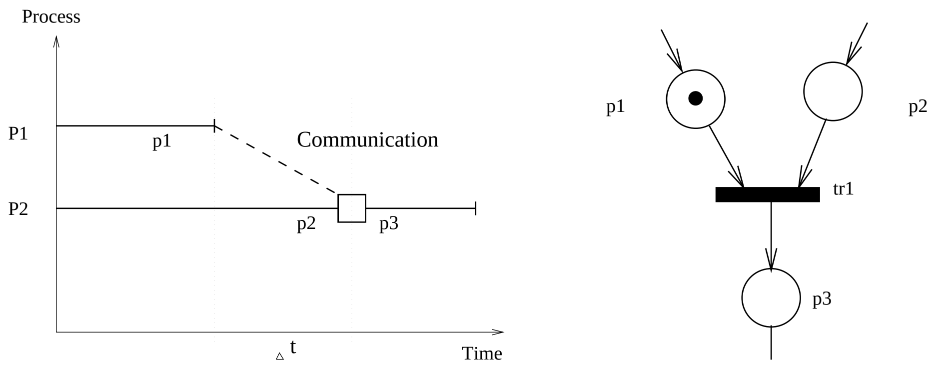

Figure 2: A system of two processes.

It is obvious that two processes $P1$ and $P2$ will be executed. After some instructions for instance process $P2$ needs some data generated by $P1$. When receiving this information, the execution of the process $P2$ will be continued and $P1$ will be terminated. The same algorithm can be represented if process $P1$ continues the execution after receiving some other information from process $P2$ which then will finish its execution. A load balancing between the processors executing $P1$ and $P2$ could be achieved if the following part after

[1]**P**arallel **V**irtual **M**achine for UNIX clusters from the Oak Ridge National Laboratory (see: Sunderam, V.S.: PVM.- in: Concurrency: Praktice and Experience, Dec. 1990, pp. 315-339, Atlanta 1990)

$p3$ will be processed on that machine which has less load ([3]) than the other. This can influence the runtime especially in the case, when the server processes (like $S1$ in figure 1) runs on a busy machine. All necessary decisions will be now obtained from the current run time situation during the program run. Such an adaption of the software requires that

- each machine is able to execute each part of the work and
- all alternatives of the execution are included in the structure of a uniform process which can be multiple started with several entry points.

3.2 Realization

Now we have the frame work to describe our load balancing approach. From our experience, in common sense the required information generated from another process and transmitted via communication procedures is not available in time.

In most cases, one process enters the communication procedure some tics earlier ($_\Delta t$). This expected time difference can be predicted using statistical estimated simulations or analysis or run time records.

Let us consider the two processes with the structure mentioned above. Let's suppose that $P1$ reaches the commmunication point at first (in case of $P2$ has this property, the considerations are the same but with changed identifiers). The time until $P2$ will finish his work can be used for calculating an optimal load balancing strategy without influencing the execution time of the whole program[2]. Figure 3 shows the Petri Net model of the needed communication procedures, if $P2$ passes $p2$ at first all necessary structures can be added symmetrically.

The functionality is the following. If $P1$ reads $Message$ at first, it contains the initial data word $< 1 >$. This will be replaced by the topical load factor of the corresponding CPU. Since $P1$ had to wait for $P2$ in each case no time will be wasted if the difference $_\Delta t$ is big enough. Passing the communication point, $P2$ gets the information about the load factor of $P1$ and can compare this information with its own. So $P2$ can decide if the process will be continued on this machine or not. Furthermore, $P1$ gets the decision via another communication and can continue its execution now. This communication is not a new overhead. Even without the load balancing described above, $P2$ has to communicate with $P1$ in each case in order to sent its final results to $P1$. Thereafter the initial value of $Message$ will be restored for following requests by the terminating process.

If we consider the expenditure of our method at all it must be mentioned that messages will be longer than without it. The decision of $P2$ about the load balancing has to be added to the message in each case.

If the parallel program does not consist of atomic processes[3], run time data values must also be transmitted. One advantage of the methods described in [7] is that parallel programs generated from Petri nets always contain only atomic processes.

The additional transfer time of the longer message and the load decision time of $P2$ can be considered as the expenditure of our method if the time $_\Delta t$ is big enough.

[2] If there is a pure multiprocessing mode; in a multitasking mode the computing time can be neglacted, if the number of processes is $>> 1$.

[3] An atomic process gets initial data while it is started and communicates with other processes after its termination when all other computation is done.

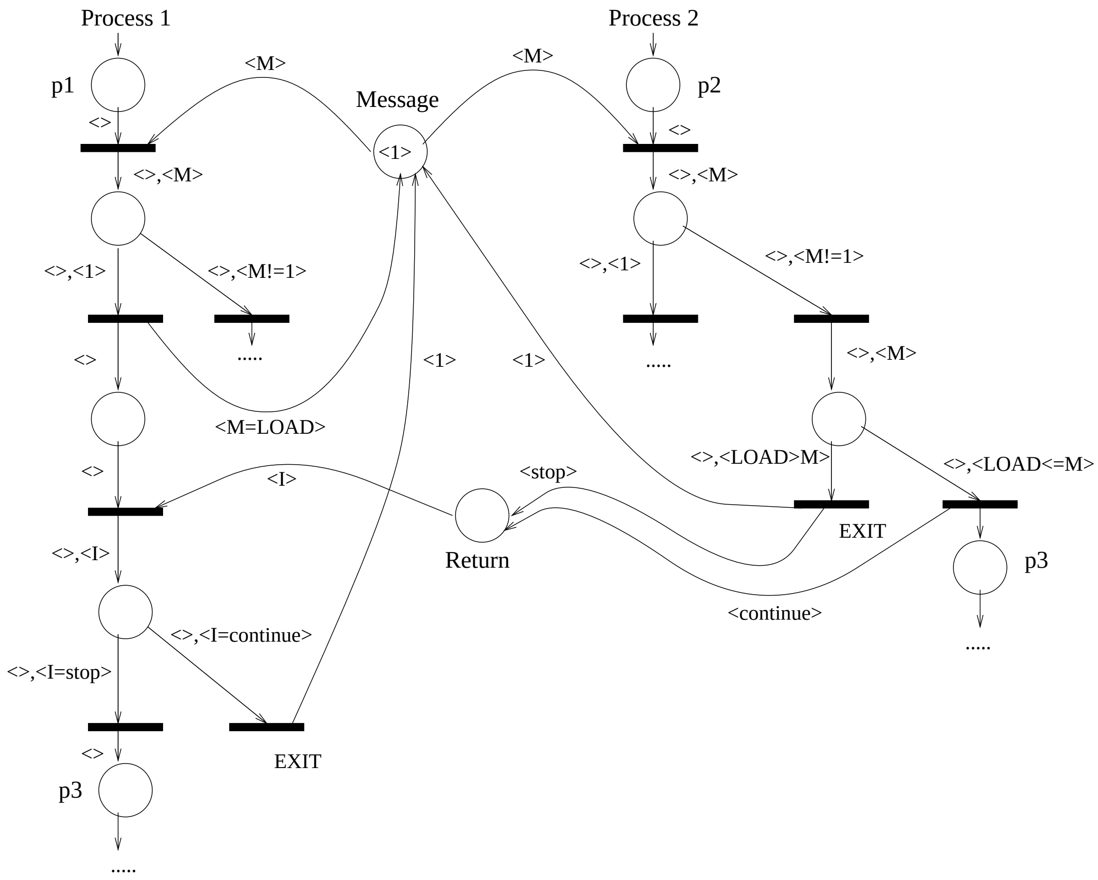

Figure 3: Introduction of the load balancing strategy.

This approach can also be generalized for three or more processes and rendezvous constructs. Now it will be inserted as a part of a source code of an existing C-program but it can be designed as a new procedure in the PVM communication library, too. This function replaces the old *pvm_send* and *pvm_recv* procedures. In addition to the arguments of the old functions, the new ones need an additional data area. It contains the values of run time data which must be sent if of the considered process hands over the further execution to the other process and will finish itself after sending the data.

Some communication points will not meet the requirements concerning $_\Delta t$ or demand the transmission of a large amount of run time data. In this case all load balancing activities can be prohibited by using the old communication functions or the old source code. Furthermore, a balancing activity can be forced by inserting the new functions with an empty message; but this results in additional execution time. Note that these exceptions or influences to the balancing are advantages of our new approach, too.

4 Conclusions

In our contribution we presented a new approach for a designer controlled load balancing for automation appliances. Therefore, we use a-priori information about the implemented software as well as run time data about the state of the system. We introduce a new kind of procedures merging necessary communication and load balancing. Therefore, expected 'blocked on receive time' of a processor during a communication can be used more efficiently for a load balancing. Furthermore, these activities can be controlled by the user by selecting only special points in dependency on a-priori process information. That is why wrong process migration can be avoided and a good run time behaviour is achieved. In future works should be considereded, if the results of common load balancing procedures can be increased while introducing special system calls which allow to block and release the process migration of the calling process in dependency on the expected time behaviour.

References

[1] M. Harchol-Balter; A.B. Downey: Exploiting Process Lifetime Distributions for Dynamic Load Balancing.-
Technical Report TR-95-021, ICSI, Berkeley, (1995)

[2] W. Brauer; W. Reisig; G. Rozenberg: Petri-Nets: Applications and Relationships to Other Models of Concurrency.-
LNCS 255, Springer-Verlag, Berlin-Heidelberg-New York, (1987)

[3] F. Burchert; M. Koch; G. Hipper; D. Tavangarian: Task Distribution in a Workstation Cluster with a Concurrent Network.-
Proceedings of APPT'95, Peking, (1995)

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

[5] H.-U. Heiss; M. Schmitz: Distributed Load Balancing Using a Physical Analogy.-
Technical Report 5/92, University of Karlsruhe, Dept. of Informatics, (1992)

[6] G. Hipper; M. Klein; D. Tavangarian: Parallel Computing in Workstation Clusters using a Concurrent Network Architecture.-
ISSM94, Washington, D.C., (1994)

[7] H. Unger; B. Däne; W. Fengler: An Approach Designing Parallel Software for Distributed Control Systems.-
13th IFAC Workshop on Distributed Computer Control Systems, Toulouse, (1995)

[8] S. White; A. Alund; V.S. Sunderam: Performance of the NAS Parallel Benchmarks on PVM based Networks.-
NASA, RNR 94-008, (1994)

[9] S. Zhou; X. Zheng; J. Wang; P. Delisle: Utopia: a load-sharing facility for large heterogeneous distributed computing systems.-
Software-Practice and Experience, 23(2), (1993)

[10] A. Barak; G. Shai; R.G. Wheeler: The MOSIX Distributed Operating System: Load Balancing for UNIX.-
Springer Verlag, Berlin, (1993)