Dynamic Load Balancing on Heterogeneous Workstation Clusters with Irregularly Fluctuating Capacity

Helmut Hlavacs

Christoph W. Ueberhuber

Institute for Applied and Numerical Mathematics,
University of Technology, Vienna
hlavacs@aurora.tuwien.ac.at,
christof@uranus.tuwien.ac.at

August 1998

AURORA TR1998-11

Abstract

A new system for simulating dynamic load balancing on interactively used heterogeneous workstation clusters is presented. Competing workload can be modeled in various ways, from simple fixed rate Poisson arrival processes, over user tracefiles to more sophisticated user behaviour graphs. Instead of running real parallel workload, the program designer builds an application model, which is then run on the simulation system. The simulator provides standard UNIX load averages and can easily be adapted and extended for special purposes. As a case study, a real parallel program is modeled and simulated.

1 Introduction

Heterogeneous workstation clusters are getting increasingly attractive for running parallel software. When running parallel programs on *interactively* used workstations, program designers have to take care not only of balancing the computational load of their program, but also of reacting to changes of the workstations' workload caused by other user programs. This redistribution of work at runtime requires *dynamic load balancing* and has been studied extensively in literature (Krommer, Ueberhuber [11], Shivaratri et al. [17], Hac, Jin [7], Burdorf, Marti [3]). In principle, there are various ways of comparing different load balancing strategies with each other. All, though, have to model the occurring workload in one way or the other.

Theoretical approaches include scalability analysis (Kumar, Gramar and Vempaty [12]) and queuing anaylsis (Wang, Morris [19]). In both techniques, workload is modeled by Poisson arrival processes in most cases.

Simulation of load balancing techniques may be performed directly by running benchmark programs on existing workstations (Arpaci, Dusseau, Vahdat [2]) or by using special simulation systems. Simulation systems might either use traces of real user sessions (Zhou [20]) or may be based on Poisson processes as well (Kunz [13]).

There are, however, various ways in modeling workstation workload. Among them are, for instance, Poisson processes (Allen [1]) with constant arrival and departure rates (Kunz [13]), Poisson processes with variable arrival rates during the day (Calzarossa, Serazzi [4] [6]), Markovian type models (Haring [9], Calzarossa, Serazzi [5]) and probabilistic context free grammars (Rhagavan, Joseph [15]).

In this paper, a simulator based on MISS-PVM (Kvasnicka [14]) is presented, which is designed to simulate different load balancing strategies for parallel programs run on interactively used, heterogeneous workstation clusters. The workload of workstations can be modeled in various ways, including birth-death processes, user session trace files and user behavior graphs. The application of the simulator is demonstrated by modeling the parallelization of the VISTA (Halama et al. [8]) ion implantation.

2 The Workstation User Simulator

The *Workstation User Simulator* (WUS) is an add-on to the *Machine Independent Simulation System for PVM3* (MISS-PVM). As with real parallel program runs, the simulated processes are started and use PVM3 (Sunderam et al. [18]) for communication. MISS-PVM, implemented as an independent layer between the user program and PVM, provides a *virtual time* depending on the process CPU usage, the speed of the interconnection network and the size of message packets sent from one process to the other. In principle, user processes can be started

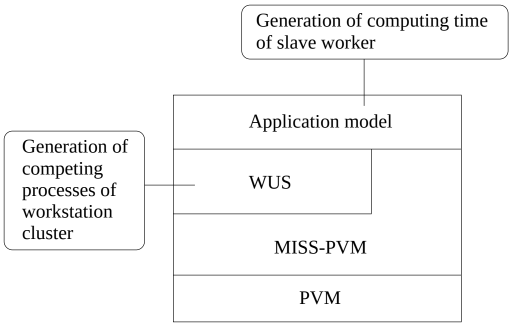

Figure 1: Structure of the simulation system.

anywhere, as long as the workstations they run on are of the same computing power. It is also possible to start all processes on one machine. When sending messages to other processes, MISS-PVM increases the virtual time according to the recently consumed CPU time, information that is provided by all standard UNIX systems. This timing information is then hooked onto the message sent. The receiver uses this time information to increase its own virtual time. The structure of the simulator system can be seen in Fig. 1. When using WUS, the real parallel application has to be replaced by an application model program, creating random CPU requests. These random requests, of course, must follow the observed statistical distribution of the real application.

These requests (say N CPU seconds) are then passed to WUS, who puts the request into its run queue and additionally produces competing processes due to the chosen workload model. After the application model has received CPU service, it again takes over control and is free to communicate with other programs via MISS-PVM or to perform the chosen load balancing strategy. The currently implemented queuing discipline is *processor sharing*, i.e. all programs receive the same amount of CPU time.

Additionally, load averages similar to the standard UNIX load averages are available, as load balancing algorithms often rely on the exponentially smoothed CPU queue length. Exponential smoothing in essence means trying to estimate the level Q of a stationary process X_t, where X_t defines the observation at time t (Schlittgen, Streitberg [16]):

$$Q_{t+1} = \alpha Q_t + (1 - \alpha) X_{t+1}$$

where Q_t is the estimate for Q at time t. The smoothing constant α defines, how many of the past observations are used for the current estimate. If the smoothing is to be done over the last N seconds, then $\alpha = N/(N + 1)$. Currently, the load averages over the last 5, 30, 60, 300 seconds (5 minutes), and 900 seconds (15 minutes) are available.

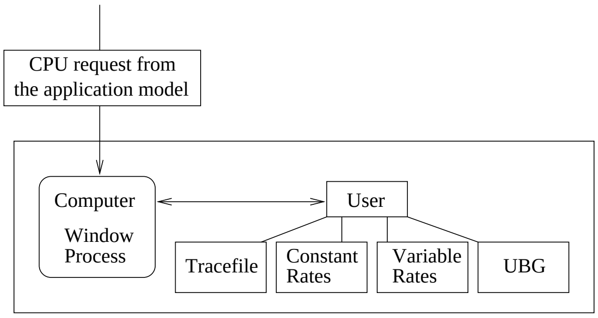

Figure 2: Structure of the Workstation User Simulator (WUS).

3 Using WUS

WUS consists of a set of C++ classes, designed to mimic a normal UNIX computer (Fig. 2). The most important class is called *Computer*. It generates and stores all other classes and interacts with the application model. The class *User* and its derivatives produce competing workload on this machine. Workload is produced by using tracefiles of real user sessions, Poisson arrival processes with fixed and variable arrival and departure rates, and user behavior graphs.

Designers of parallel programs wishing to use WUS to test load balancing strategies first have to sample statistical data of the CPU requests of their real parallel applications. Using this data, a statistical application model then has to be created. An application model program frame looks like the following example.

```
do communication or initialization
pComputer = new Computer( Workload model );

while( Loop ) {
    runtime = GetRandomRuntime();
    pComputer->RunProcess( runtime );
    loadavg = pComputer->LoadAverage(n);
    do communication or load balancing
}
collect results
```

Initialization is done by creating an instance of the class *Computer*. Here, the type of workload model is passed to the *Computer* class, which in turn creates the necessary *User* classes. Processor requests are passed to the computer by calling the memberfunction `RunProcess()`. Load averages then can be retrieved by calling `LoadAverage()` and can be used to perform load balancing. Communication to other processes is carried out by normal PVM3 calls (which are replaced by MISS-PVM calls).

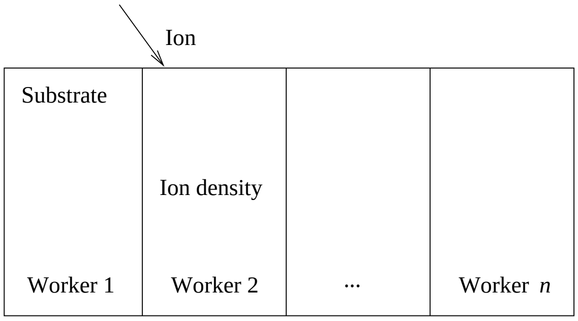

Figure 3: VISTA ion implantation.

4 Simulating the VISTA Ion Implantation

The VISTA (Halama et al. [8]) ion implantation program, developed at the Institute for Microelectronics at the Technical University of Vienna, is a Monte Carlo simulator computing the endpoints of ions shot into a substrate. The resulting ion density is needed to predict the electrical behavior of the semiconductor. The application requires large amounts of computation time and thus has been parallelized to run on institute's workstation cluster. The workstations are used by interactive users, as well as by other parallelized programs, like large compilation jobs. Currently, no load balancing is implemented, causing the implantation program to run out of balance quite frequently. Fig. 3 shows the implantation process.

For the parallel version, the region is split into n parts. If ions leave their segment they are sent to the owner of the neighbor segment. One important fact limits the parallelization of the implantation. In the crystalline case, the substrate itself changes due to the ion paths. After the implantation of at most 500 ions, the workers must synchronize.

In order to reflect the heterogeneity of the VISTA workstation cluster, each available workstation has been tested using the *Whetstone* floating-point benchmark. Additionally, a user process has logged the CPU and memory demands of all processes run on the workstation cluster for two months, resulting in over 500 tracefiles.

The CPU demands per ion have been modeled by statistical distributions. For the overall ion time, a 3-stage gamma distribution function $G(t)$ has been fitted by using the non-linear minimization capabilities of the statistical package R (Ihaka [10]). The task was to find the set of parameters minimizing the squared sum of differences at the jump points to the cumulative distribution function $S_n(\cdot)$ provided by the data

$$SE = \sum_{x_{jump\ point}} (G(x) - S_n(x))^2.$$

Additionally, if ions are sent to a neighbor, the CPU time spent in their segment has been modeled for over 50 different segment sizes. This is important, as parts of segments will be shifted to neighbors in case of load balancing activities. Each of these distributions consist of a *2-stage hyperexponential distribution* with distribution function

$$F(t) = \alpha(1 - e^{-t/\lambda_1}) + (1 - \alpha)(1 - e^{-t/\lambda_2}).$$

Here, fitting has been done by using a mixture of the method of moments (Allen [1]) and non-linear minimization. By computing the first two statistical moments

$$E[X] = \alpha\lambda_1 + (1 - \alpha)\lambda_2 \equiv a$$
$$E[X^2] = 2\alpha\lambda_1^2 + 2(1 - \alpha)\lambda_2^2 \equiv b$$

the distribution means λ_1 and λ_2 can be related to α by

$$\lambda_1 = \frac{a - (1 - \alpha)\lambda_1}{\alpha}, \qquad \lambda_2 = a \pm \sqrt{a^2 - \frac{2a^2 - \alpha b}{2(1 - \alpha)}}.$$

Thus, only α lying in the interval $[0, 1]$ has to be fitted to the data. Each distribution has been evaluated by the Kolmogorov-Smirnov goodness-of-fit test. Most have been accepted with high probability, rejections at least show significance at a one percent level. Parameters for segments not being considered are linearly interpolated.

5 Simulation Results

Simulation was carried out by starting ten processes, each using a tracefile of a real workstation day. As workstations, 2 of the fastest machines (320M Whetstones/second) and 8 of the slowest machines (75M Whetstones/second) were chosen. The data size describing the segment structure was set to 50 MB, here anything between a few MBs and 1 GB is possible. The simulation start time was set to 4 pm, thus increasing the simulation time accordingly. At 4 pm the master process sends load requests to all processes, then chooses k workstations with the best performance/load ratio. Fig. 4 shows the simulation results. Load balancing is carried out at each synchronization point. The master receives load information of its workers and starts the load balancing mechanism, in case the load imbalance exceeds some threshold. Load balancing is carried out by shifting the segment limits and sending the according segment data to other workers. The lower limit denotes the case, where no competing processes are started, thus yielding the full computational power to the application model.

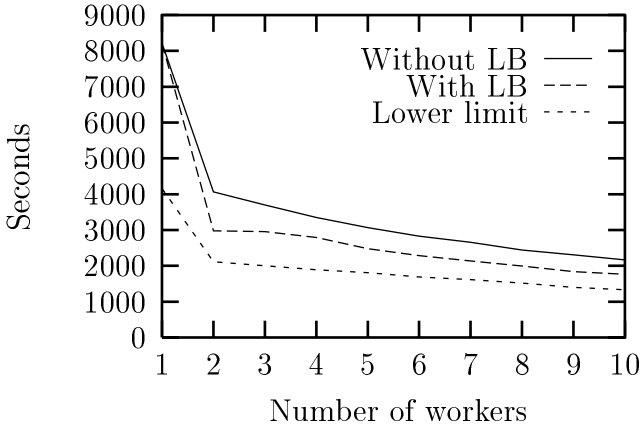

Figure 4: Effect of load balancing (LB).

6 Future Work

The workstation tracefiles indicate large fluctuations of workload during the day. Especially the faster machines are more likely to get overloaded. Load balancing algorithms to be yet developed must be able to predict the future workload, at least to some degree, to avoid unnecessary work shifts.

Also, the simulator itself will be extended to accept not only application models but real application programs as well. Additional features like other queueing disciplines including the UNIX nice level will be provided.

7 Conclusion

In this paper a simulator designed especially for the development of load balancing algorithms on interactively used heterogeneous workstation clusters has been introduced. In this simulator various ways of describing workstation workload can be used, reaching from simple fixed arrival/departure rates up to sophisticated user behavior graphs. The newly developed simulation system will be an important tool for evaluating load balancing algorithms in various applications, amongst them the VISTA ion implantation.

References

[1] Allen A.O., *Probability, Statistics and Queuing Theory*, Academic Press Inc, Orlando San Diego New York Austin London, 1990.

[2] Arpaci R.H., Dusseau A.C., Vahdat A.M., *The Interaction of Parallel and Sequential Workload on a Network of Workstations*, in SIGMETRICS '95, ACM, 1995.

[3] Burdorf C., Marti J., *Load Balancing Strategies for Time Warp on Multi-User Workstations*, The Computer Journal 36-2 (1993), pp. 168-176.

[4] Calzarossa M., Serazzi G., *A Characterization of the Variation in Time of Workload Arrival Patterns*, IEEE Transactions on Computers C-34-2 (1985), pp. 156-162.

[5] Calzarossa M., Serazzi G., *System Performance with User Behaviour Graphs*, Performance Evaluation 11 (1990), pp. 155-164.

[6] Calzarossa M., Serazzi G., *Workload Characterization: A Survey*, Proceedings of the IEEE 81-8 (1993), pp. 1136-1150.

[7] Hac A., Jin H., *Dynamic Load Balancing in a Distributed System Using a Sender Initiated Algorithm*, J. Systems Software 11 (1990), pp. 79-94.

[8] Halama S. et al, *The Viennese Integrated System for Technology CAD Application*, in Technology CAD Systems (F. Fasching, S. Halama, S. Selberherr, eds), Springer Vienna 1993, pp. 197-236.

[9] Haring G., *On stochastic models of interactive workloads"*, in PERFORMANCE '83 (Agrawala A.K., Tripathi S.K. eds), North Holland Amsterdam, 1983, pp. 345-361.

[10] Ihaka R., *R: Past and Future History*,
URL: http://www.stat.math.ethz.ch/CRAN/

[11] Krommer A., Ueberhuber C., *Dynamic Load Balancing - An Overview*, Technical Report ACPC/TR 92-2, Austrian Center for Parallel Computation, Vienna, 1992.

[12] Kumar V., Grama A.Y., Vempaty N.R., *Scalable Load Balancing Techniques for Parallel Computers*, Journal of Parallel and Distributed Computing 22 (1994), pp. 60-79.

[13] Kunz T., *The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme*, IEEE Transactions on Software Engineering 17-7 (1991), pp. 725-730.

[14] Kvasnicka D., *Developing Architecture Adaptive Algorithms using Simulation with MISS-PVM for Performance Prediction*, 11th ACM Conference on Supercomputer, Vienna, Austria.

[15] Raghavan S.V., Joseph P.J., *Workload Models for Multiwindow Distributed Environments*, in Quantitative Evaluation of Computing and Communication Systems (Beilner H., Bause F., eds), Springer Heidelberg, 1995.

[16] Schlittgen R., Streitberg B., *Zeitreihenanalyse*, R. Oldenbourg Verlag Muenchen Wien, 1995.

[17] Shivaratri N.G., Krueger P., Singhal Mukesh, *Load Distributing for Locallay Distributed Systems*, Cmputer 12 (1994), pp. 33-44.

[18] Sunderam V.S., Geist G.A., Dongarra J., Manchek R., *The PVM concurrent computing system: Evolution, experiences and trends*, Parallel Computing 20-4 (1994).

[19] Wang Yung-Terng, Morris R.J.T, *Load Sharing in Distributed Systems*, IEEE Transactions on Computers C-34-3 (1985), pp. 204-217.

[20] Zhou S., *A Trace-Driven Simulation Study of Dynamic Load Balancing*, Tech. Rept No. UCB/CSD 87/305, September 1986.