Simulating Load Balancing on Heterogeneous Workstation Clusters*

Helmut Hlavacs[1] and Christoph W. Ueberhuber[2]

[1] Institute for Applied Computer Science and Information Systems, University of Vienna
hlavacs@ani.univie.ac.at
[2] Institute for Applied and Numerical Mathematics, Technical University of Vienna
christof@uranus.tuwien.ac.at

Abstract. A new system for simulating dynamic load balancing on heterogeneous workstation clusters is presented. Competing workload can be modeled in various ways, from simple to sophisticated. 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.

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 [12], Shivaratri et al. [19], Hac, Jin [9], Burdorf, Marti [4]). 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 [13]) and queuing analysis (Wang, Morris [22]). 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 [23]) or may be based on Poisson processes as well (Kunz [14]).

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 [14]), Poisson processes with variable arrival rates during the

* This work was supported by the Austrian Science Fund (*Österreichischer Fonds zur Förderung der wissenschaftlichen Forschung*).

day (Calzarossa, Serazzi [5] [7]), Markovian type models (Haring [10], Calzarossa, Serazzi [6]) and probabilistic context free grammars (Rhagavan, Joseph [17]).

In this paper, a simulator based on MISS-PVM (Kvasnicka, Ueberhuber [15] [16]) is presented, which is designed to simulate different load balancing strategies for parallel programs run on interactively used, heterogeneous workstation clusters. Using this new simulation tool, the workload of workstations can be modeled in various ways, including Poisson processes, user session trace files and user behavior graphs (UBGs). The application of the new simulator is demonstrated by modeling the parallelization of the ion implantation module (Bohmayr, Burenkov, Lorenz, Ryssel, Selberherr [3]) which is part of the technology CAD system VISTA (Strasser, Pichler, Selberherr [20], Grasser et al. [8]).

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. [21]) 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, the simulated user processes can be started on any single or multi-processor computer.

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 whole simulation system can be seen in Fig. 1.

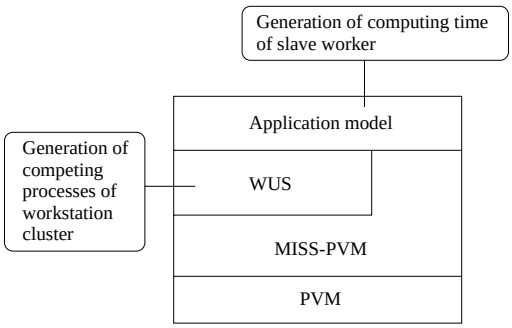

Fig. 1. Structure of the simulation system.

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. The synthetically generated CPU requests (for instance, one request may refer to T CPU seconds) are then passed to WUS, which puts the request into its run queue and additionally produces competing processes due to the chosen workload model. After the application model has consumed its requested CPU time, 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 processes 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, for which observations $\ldots, X_{t-1}, X_t, X_{t+1}$ at times $\ldots, t-1, t, t+1$ are available (Schlittgen, Streitberg [18]):

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

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

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 produces competing workload on this virtual machine. Workload is produced by using trace files of real user sessions, Poisson arrival processes with fixed and variable arrival and departure rates, and user behavior graphs.

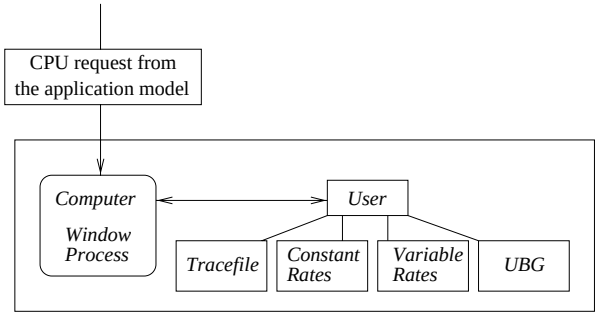

Fig. 2. Structure of the Workstation User Simulator (WUS).

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 application. Using this sampling data, a statistical application model then has to be created. An application model program frame may look 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*. The type of workload model is passed to the *Computer* class, which in turn creates the necessary *User* classes. Processor requests are passed to *Computer* by calling the member function `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).

4 Simulating the VISTA Ion Implantation

VISTA (Strasser, Pichler, Selberherr [20], Grasser et al. [8]), developed at the Institute for Microelectronics at the Technical University of Vienna, is a framework for the design and simulation of process steps involved in semiconductor production. VISTA includes an ion implantation program (Bohmayr et al. [3]), which is based on a Monte Carlo simulator computing the endpoints of ions shot into a substrate. The resulting ion density is needed later on to predict the electrical behavior of the investigated semiconductor.

Monte Carlo ion implantation requires large amounts of computation time and thus has been parallelized to run on the institute's workstation cluster. The workstations of this cluster are not dedicated to ion implantation jobs but are also used by interactive users, as well as by other parallelized programs, like large compilation jobs. Currently, no dynamic 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 of the Monte Carlo ion implantation, currently under development at the Institute for Microelectronics, the whole semiconductor region is split into n parts (see Fig. 3). If ions leave their segment they are sent to the owner of the neighboring segment. One important fact limits the parallelization of the ion implantation. In the crystalline case, the penetrating ions

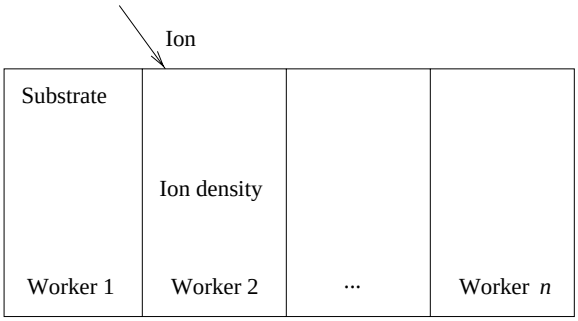

Fig. 3. VISTA ion implantation.

change the properties of the substrate. 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 whole workstation cluster for two months, resulting in over 500 trace files.

The CPU demands per ion have been modeled by statistical distributions. For the overall ion time, a 3-stage gamma distribution function G has been fitted by using the non-linear minimization capabilities of the statistical package R (Ihaka [11]). The task was to find the set of parameters minimizing

$$SE = \sum_{x_j} (G(x_j) - S_n(x_j))^2,$$

i. e., the squared sum of differences between the observed cumulative distribution function S_n and the model distribution function G. Differences are taken at the jump points x_j of the distribution function S_n.

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 $\alpha \in [0,1]$ 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 have been linearly interpolated.

5 Simulation Results

Simulation was carried out by starting ten processes, each using a trace file of a real workstation day. From the workstation cluster, two of the fastest machines (320 MWhetstones/second) and eight of the slowest machines (75 MWhetstones/second) were chosen. The data size describing the segment structure was set to 50 MB (anything between a few MBs and 1 GB would have been 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 assigning the full computational power to the ion implantation calculations, i. e., the application model under consideration.

6 Future Work

The workstation trace files 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 queuing 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

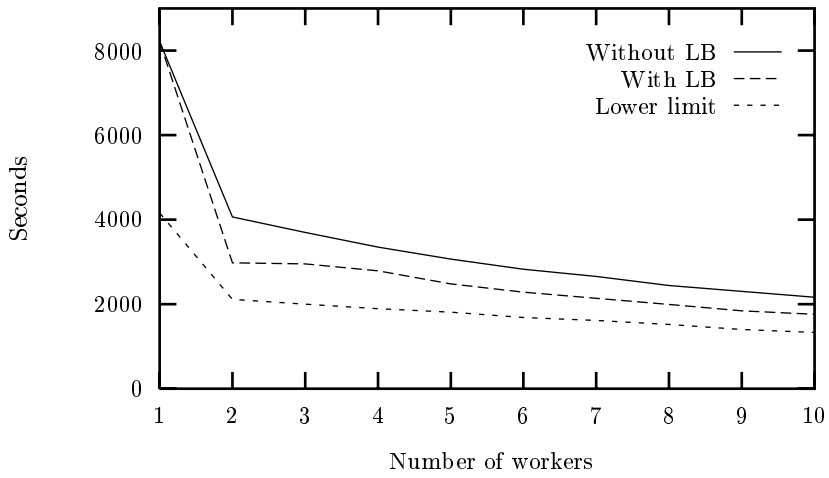

Fig. 4. Effect of load balancing (LB).

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 Monte Carlo ion implantation module of VISTA.

Acknowledgments

We would like to thank Siegfried Selberherr, Erasmus Langer, Mustafa Radi and Andreas Hössinger (Institute for Microelectronics, Technical University of Vienna) for their cooperation.

Additionally, we would like to acknowledge the financial support of the Austrian Science Fund FWF.

References

1. Allen A.O., *Probability, Statistics and Queuing Theory*, Academic Press, Orlando, 1990.
2. Arpaci R.H., Dusseau A.C., Vahdat A.M., *The Interaction of Parallel and Sequential Workload on a Network of Workstations*, Performance Evaluation Review 23-1 (1995), pp. 267-278.
3. Bohmayr W., Burenkov A., Lorenz J., Ryssel H., Selberherr S., *Monte Carlo Simulation of Silicon Amorphization During Ion Implantation*, Proceedings SISPAD 96 Conf., (2.-4. September 1996, Tokyo), pp. 17-18.
4. Burdorf C., Marti J., *Load Balancing Strategies for Time Warp on Multi-User Workstations*, The Computer Journal 36-2 (1993), pp. 168-176.
5. 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.

6. Calzarossa M., Serazzi G., *System Performance with User Behavior Graphs*, Performance Evaluation 11 (1990), pp. 155-164.
7. Calzarossa M., Serazzi G., *Workload Characterization: A Survey*, Proceedings of the IEEE 81-8 (1993), pp. 1136-1150.
8. Grasser T. et al, *VISTA Status Report*, Technical Report AURORA TR1997-16, Technical University of Vienna (1997).
9. Hac A., Jin H., *Dynamic Load Balancing in a Distributed System Using a Sender Initiated Algorithm*, J. Systems Software 11 (1990), pp. 79-94.
10. 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.
11. Ihaka R., *R: Past and Future History*, http://www.stat.math.ethz.ch/CRAN/
12. Krommer A., Ueberhuber C., *Dynamic Load Balancing—An Overview*, Technical Report ACPC/TR 92-2, Austrian Center for Parallel Computation, Vienna, 1992.
13. 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.
14. 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.
15. Kvasnicka D., Ueberhuber C.W., *Simulating Architecture Adaptive Algorithms with MISS-PVM*, Technical Report AURORA TR1997-16, Technical University of Vienna (1997).
16. Kvasnicka D., Ueberhuber C.W., *Developing Architecture Adaptive Algorithms using Simulation with MISS-PVM for Performance Prediction*, Conference Proceedings of the 1997 ACM/SIGARCH International Conference on Supercomputing, Vienna, Austria, July 7-11, 1997, pp. 333-339.
17. 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.
18. Schlittgen R., Streitberg B., *Zeitreihenanalyse*, R. Oldenbourg Verlag Muenchen, 1995.
19. Shivaratri N.G., Krueger P., Singhal Mukesh, *Load Distributing for Locally Distributed Systems*, Computer 12 (1994), pp. 33-44.
20. Strasser R., Pichler Ch., Selberherr S., *VISTA—A Framework for Technology CAD Purposes*, Proceedings European Simulation Symposium, (19.-22. October 1997, Passau), pp. 445-449.
21. Sunderam V.S., Geist G.A., Dongarra J., Manchek R., *The PVM concurrent computing system: Evolution, experiences and trends*, Parallel Computing 20-4 (1994), pp. 531-545.
22. Wang Yung-Terng, Morris R.J.T, *Load Sharing in Distributed Systems*, IEEE Transactions on Computers C-34-3 (1985), pp. 204-217.
23. Zhou S., *A Trace-Driven Simulation Study of Dynamic Load Balancing*, Technical Report UCB/CSD 87/305 (1986).