1998

Dynamic Management Of Heterogeneous Resources

Jerrell Watts
Syracuse University, Scalable Concurrent Programming Laboratory, jwatts@scp.syr.edu

Marc Rieffel
Syracuse University, Scalable Concurrent Programming Laboratory, marc@scp.syr.edu

Stephen Taylor
Syracuse University, Scalable Concurrent Programming Laboratory, steve@scp.syr.edu

DYNAMIC MANAGEMENT OF HETEROGENOUS RESOURCES

Jerrell Watts, Marc Rieffel and Stephen Taylor
Scalable Concurrent Programming Laboratory
Syracuse University
Syracuse, NY 13244-4100
e-mail: {jwatts, marc, steve}@scp.syr.edu

Keywords: concurrent computing, dynamic load balancing, heterogeneous environments

ABSTRACT

This paper presents techniques for dynamic load balancing in heterogeneous computing environments. That is, the techniques are designed for sets of machines with varying processing capabilities and memory capacities. These methods can also be applied to homogenous systems in which the effective compute speed or memory availability is reduced by the presence of other programs running outside the target computation. To handle heterogeneous systems, a precise distinction is made between an abstract quantity of work, which might be measured as the number of iterations of a loop or the count of some data structure, and the utilization of resources, measured in seconds of processor time or bytes of memory, required by that work. Once that distinction is clearly drawn, the modifications to existing load balancing techniques are fairly straight-forward. The effectiveness of the resulting load balancing system is demonstrated for a large-scale particle simulation on a network of heterogeneous PC's, workstations and multiprocessor servers.

INTRODUCTION

The use of networks of personal computers, workstations and symmetric multiprocessors as a computing platform requires improved dynamic load balancing techniques. Unlike traditional multicomputers, such as the Cray T3D/E or the Intel Paragon, the computers in a typical network are often not of the same processing performance nor do they have the same available memory. As a result, most existing techniques for dynamic load balancing, which consider computing resources to be homogeneous, are insufficient (Watts and Taylor 1998b; Willebeek-LeMair and Reeves 1993; Xu and Lau 1997). Those techniques which have addressed heterogeneous environments have primarily been geared towards manager-worker or task-queue problems (Bowen *et al.* 1992; Li and Dorband 1997; Lin and Raghavendra 1992; Zhou *et al.* 1993) rather than the single-program, multiple-data (SPMD) style of scientific computations (Cap and Strumpen 1993; Henriksen and Keunings 1993). Our work addresses load balancing on heterogeneous systems for large-scale scientific simulations. These techniques involve relatively simple modifications to existing methods already in the literature. Consequentially, they can be incorporated into many existing load balancing systems, allowing users to leverage a wider variety of machines for a computation than previously possible. Experiments conducted on a network composed of PC's running Windows NT, together with workstations and multiprocessor servers running various versions of Unix, demonstrate the effectiveness of these techniques.

Before one can address the problem of load balancing in a heterogeneous environment, it is necessary to make clear distinctions among certain terms. When dealing with homogenous computing environments, notions of "load," "work," "utilization," and "runtime" are often used interchangeably. This is perfectly acceptable because an abstract quantity of work, say 100 iterations of a fluid flow solver over a 100,000-cell grid partition, presumably requires the same execution time on every machine. In heterogeneous environments this is definitely not the case. If a problem is characterized by abstract, algorithmic considerations such as the number of operations or the count of data structures, this will translate differently into processor or memory usage depending on the particular machine on which the problem is run. So, a distinction must be made between the quantities which are invariant across a set of computers and those which vary according to the computer in question. We refer to the former, abstract algorithmic quantities as the *load* of a computer. For example, in a particle simulation, a partition of the problem may contain 125,000 particles. The load for that region could be taken to be 125,000. On one machine, processing those particles might require five compute seconds per iteration and on another machine require ten seconds per iteration. We refer to this variant quantity as the *utilization.* The degree to which a particular computer is utilized by a certain load is determined by that computer's *capacity.*

BASIC METHODOLOGY

The heterogeneous dynamic load balancing techniques presented here are a modification to the homogeneous load balancing framework presented previously (Watts and Taylor 1998b). That framework achieves load balance by dynamically relocating tasks from computer to computer, as well as increasing or decreasing the number of tasks by dynamically dividing or merging them. The process occurs in six phases:

1. **Load measurement:** The loads of the tasks on a computer are determined, and those loads are summed to yield the total load at each computer.
2. **Profitability determination:** Once the loads of the computers are determined, the decision is made to continue if the improvement possible with load balancing justifies its cost.
3. **Load transfer calculation:** The ideal quantities of load to transfer between computers are calculated.
4. **Task selection:** Tasks are selected for exchange between computers to meet the ideal transfer quantities. This process may be repeated several times until transfer quantities are adequately met, and tasks may be divided to increase the options available.
5. **Task migration:** Tasks' data structures are transferred to their new locations.
6. **Granularity adjustment:** Tasks may be divided to make better use of multiple processors sharing memory or merged to reduce system overhead.

This decomposition allows different components of the load balancing framework to be replaced in a plug-and-play fashion, providing the ability to customize it for particular applications or computing environments.

HETEROGENEOUS METHODOLOGY

Each of the above phases must be modified for use in heterogeneous environments. As stated above, the modifications primarily entail the distinction between load and utilization, as well as the incorporation of computer capacities.

Load measurement

In determining the load of a task, there are two options. One is to use abstract, algorithmic quantities such as the number of operations or data structures. If the load of task j is taken to be l_j, then the load of computer i is $L_i = \sum_{j \in T_i} l_j$, where T_i is the set of tasks mapped to that computer. In that case, the utilization of computer i is given by $U_i = \frac{L_i}{C_i}$, where C_i is the computer's capacity. Similarly, the utilizations due to the tasks are given by $u_j = \frac{l_j}{C_{M(j)}}$, where $M(j)$ is the computer to which task j is mapped.

As a second option, one can measure the utilization using system facilities. For example, one might use system calls to get the CPU time or amount of memory used by a task. In that case, one simply reverses the above formulas to calculate the abstract loads: $l_j = C_{M(j)} u_j$ gives the load of a task. The resulting task loads and utilizations are summed to yield the computers' total loads and utilizations, respectively.

Of course, in both of the above cases, it is assumed that one knows the resource capacity of a given computer. The capacity can be determined in a number of ways. If the capacity measured is processing speed, a benchmarking program—possibly the target application with a smaller test problem—can be used to determine the relative speeds of various machines. These offline performance numbers, along with other statistics, such as the machines' memory capacities, can be put into a file which is read at the start of the computation. A third method is to use both invariant algorithmic quantities and system-measured utilization numbers. For example, one might use the number of cells in a grid partition as well as the CPU time required to process those cells. By dividing the former by the latter, one can calculate the capacity of the system dynamically during execution.

If the application is run in the presence of external programs which also compete for system resources, the capacities must be adjusted. For example, one might divide the processing capacity for a given machine by its "load average," which is a measure of the average number of processes competing for CPU time.

Profitability determination

In a heterogeneous environment, determining whether load balancing is worthwhile is substantially the same as in the heterogeneous case. The degree of load balance is given by

$$eff = \frac{U_{\text{avg}}}{U_{\max}}$$

where U_{avg} and $U_{\max}$ are the average and maximum utilizations over the computers, respectively.

Load balancing would be considered whenever the efficiency falls below a user-specified minimum of $eff_{\min}$. If one is attempting to balance the run time of the machines, load balancing should be undertaken if the time required for load balancing is exceeded by the improvement in execution time that would result from a better load distribution. If one

is attempting to balance some other quantity, such as memory, a possible criterion would be to load balance whenever the physical memory capacity of any machine is exceeded, since the resulting page swapping will probably severely degrade the application's performance.

Load transfer calculation

The step most affected by consideration of heterogeneous systems is that of ideal load transfer calculation. The literature contains a number of algorithms for calculating how much load to transfer among computers, including the hierarchical balancing method, the generalized dimensional exchange, and heat diffusion methods (Boillat 1990; Cybenko 1989; Heirich and Taylor 1995; Watts and Taylor 1998b; Willebeek-LeMair and Reeves 1993; Xu and Lau 1997).

Before discussing the specifics of the modified algorithms, it is important to reiterate the fact that these are *load* transfer calculation algorithms. However, the goal of these algorithms is not necessarily to balance the loads of the computers in the sense of making them equal. Instead, the algorithms should reassign load in such a way that the load at each computer is proportional to that computer's capacity. In other words, the algorithms balance the computers' utilizations by determining the ideal redistribution of their loads.

The hierarchical balancing method (HBM) is a recursive approach in which the computers are divided into two partitions, and the total load is calculated for each partition. The load transfer from the first partition to the second is that necessary to make their respective loads per computer equal

$$\Delta L_{1,2} = \frac{P_2 L_1 - P_1 L_2}{P_1 + P_2}$$

where L_1 and L_2 are the loads of the two partitions of computers, and P_1 and P_2 are the number of computers in each partition.

In the case of a heterogeneous system, we seek to establish that the load *per capacity* (i.e., the utilization) is equal in each partition. Thus, the transfer becomes

$$\Delta L_{1,2} = \frac{C_2 L_1 - C_1 L_2}{C_1 + C_2}$$

where C_1 and C_2 are the total capacities of the two partitions. In both the homogeneous and heterogeneous cases, the two partitions are then recursively divided and balanced independently.

In the generalized dimensional exchange, the "links" between adjacent computers are colored so that no computer has two links of the same color. The colors are iterated over, and a computer transfers load to or from its neighbor along the link of each color until an adequately balanced state is reached. The load transfer accumulation between two computers i and j at step k is

$$\Delta L_{i,j}^{(k)} = \Delta L_{i,j}^{(k-1)} + \lambda(L_i^{(k-1)} - L_j^{(k-1)})$$

where λ is a constant between 0 and 1. For a detailed discussion of the selection of λ see (Xu and Lau 1997). Note that the loads of the computers are adjusted to account for previous load transfers.

For the heterogeneous case, the transfer must be weighted to account for the relative capacities of the two computers. The resulting transfer iteration becomes

$$\Delta L_{i,j}^{(k)} = \Delta L_{i,j}^{(k-1)} + \lambda\left(\frac{C_j L_i^{(k-1)} - C_i L_j^{(k-1)}}{C_i + C_j}\right)$$

Finally, the heat diffusion algorithm can be modified to accommodate heterogeneous systems. Most presentations of heat diffusion solve the underlying partial differential equation using explicit methods (Boillat 1990; Cybenko 1989; Willebeek-LeMair and Reeves 1993; Xu and Lau 1997). The problem with these methods is that they are unstable for large time steps. To solve this problem, implicit methods can be used (Heirich and Taylor 1995). Figure 1 gives the implicit heterogeneous heat diffusion algorithm as executed by a particular computer i. (Unfortunately, there is insufficient space to present the full derivation of this somewhat complex algorithm.) In that figure, N_i is the set of network neighbors of computer i; α is the accuracy of the algorithm and is typically set to be $1 - eff_{\min}$.

Task selection

Once the ideal load transfer quantities have been calculated, the next step is to determine which tasks to transfer or exchange among computers to meet those quantities. The problem of selecting which tasks to transfer is essentially the subset sum problem, for which polynomial time approximation algorithms exist. An alternative approach, however, is to formulate the problem in terms of the 0-1 knapsack problem, allowing transfer cost to be factored into the decision process (Watts and Taylor 1998a). For example, one might wish to reduce the effect that task transfer has on an application's communication locality.

The selection process is repeated multiple times until the transfer quantities are adequately achieved or no further progress is made. If the tasks are too coarse-grained, it may be impossible to balance the load. If the user has provided the appropriate support routines, one can divide the data structures of a given task to produce multiple smaller tasks. One way to select tasks for division is to choose a threshold and divide any tasks whose utilizations exceed that threshold. If the division of tasks results in a better, but still inadequate load balance, the threshold would then be lowered so that more tasks would be divided. This would

diffuse(…)

$\Delta L_{i,j} := 0$ for all $j \in N_i$

$$D_i := 1 + \alpha \sum_{j \in N_i} \frac{C_j}{C_i + C_j}$$

$$T_{i,j} := \alpha \frac{C_i}{C_i + C_j}$$

$$m := \left\lceil \frac{\ln \alpha}{\ln \left[\max_{i'} \left(D_{i'}^{-1} \sum_{j' \in N_{i'}} T_{i',j'} \right) \right]} \right\rceil$$

while $eff < eff_{\min}$ do

$L_i^{(0)} := L_i$

for $k := 1$ to m do

$$L_i^{(k)} := D_i^{-1} \left(L_i^{(0)} + \sum_{j \in N_i} T_{i,j} L_j^{(k-1)} \right)$$

end for

$$\Delta L_{i,j} := \Delta L_{i,j} + D_i^{-1} T_{i,j} \left(\frac{C_j}{C_i} L_i^{(0)} - L_j^{(m-1)} \right) \text{for all}$$

$j \in N_i$

$L_i := L_i^{(m)}$

end while

end diffuse

Figure 1: Heterogeneous diffusion algorithm.

continue until an adequate load balance is achieved, until no benefit results from finer granularity, or until further task divisions are impossible, possibly due to algorithmic considerations.

Task migration

A task can be transferred from one computer to another if the user provides three functions—one function to write the state of the task to the network, another function to read the task's state from the network, and a third function to continue execution of the computation. The transfer of a task from one computer to another is complicated in the heterogeneous case by the fact that the underlying data structures may be represented differently on different computers. For example, integers may be 32 or 64 bits in length and may be stored in either big-endian (most significant byte first) or little-endian (least significant byte first) fashion. The communication layer must thus provide conversion between the data types of the respective machines.

Note that the tasks with which a migrated task communicates must be notified of the task's new location. Moreover, if the application is not quiescent at the time of load balancing, it may be necessary to forward messages to the tasks' new locations. Ideally this process would be accomplished by the underlying concurrent programming library and would use completely local, asynchronous protocols to re-establish communication (Taylor *et al.* 1996).

Granularity adjustment

Once tasks have arrived at their new computers, it may be beneficial to increase or reduce the number of tasks on a given computer. For example, on a computer with more than one processor, there may be too few tasks to use all of the processors. In that case, tasks whose utilizations exceed the average computer utilization should be divided. Specifically, divide a task if its utilization exceeds $\frac{U_{\text{avg}}}{eff_{\min}}$, where U_{avg} is the average utilization per computer and $eff_{\min}$ is the minimum desired computational efficiency.

RESULTS

The heterogeneous load balancing framework described above was applied to a large-scale concurrent particle simulation (Rieffel *et al.* 1997). This application uses a technique called direct simulation monte carlo (DSMC). The DSMC method solves the Boltzmann equation by simulating individual particles. Since it is impossible to simulate the actual number of particles in a realistic system, a smaller number of macroparticles are used, each of which represents a large number of real particles. The simulation of millions of these macroparticles is made practical by decoupling their interactions. First, the space through which the particles move is divided into a grid. Particle collisions are considered only for those particles within the same grid cell. Furthermore, the collisions are not detected by path intersection but are instead determined by a stochastic model whose inputs include the relative velocities of the particles in question. Statistical methods are used to recover macroscopic properties such as temperature and density.

The DSMC technique described above was implemented using the Scalable Concurrent Programming Library (SCPlib). This library has been applied to a variety of large-scale industrial simulations and is portable to a wide range of platforms. On each of these platforms, the library provides a common set of low-level functionality, including

message-passing, thread management, synchronization, I/O, and performance monitoring.

The SCPlib programming model is based on the concept of a concurrent graph of communicating tasks (Taylor *et al.* 1996). A task is comprised of a thread of execution, a set of communication ports and the user's state. In the case of the DSMC application, each task is a partition of the grid. The thread executes the DSMC algorithm over that partition, transferring particles to and from neighboring partitions via the communication ports. These communication ports automatically convert basic data types between different architectures, and they effectively hide the mapping of tasks to computers. Because of this, tasks can be relocated during the course of computation for the purpose of load balancing. The load balancing framework in SCPlib uses the heat diffusion algorithm for transfer quantity calculation, cost functions to improve communication locality during task selection, and finally, facilities for task division to assist in the task selection and granularity adjustment phases.

The DSMC application was applied to a 54,000-cell box grid containing 432,000 particles. This problem was partitioned uniformly to run on a network of 10 PC's, workstations and multiprocessor servers connected via a 10/100 Mbit/sec Ethernet switch. This network included single- and dual-processor Dell PC's, a Silicon Graphics Origin 200 two-processor server, two Indigo 2 and three Indy workstations, a Sun SparcServer and a Digital Equipment AlphaStation. Included in that list are machines with both 32-and 64-bit words as well as big- and little-endian byte orderings. These machines are described in greater detail in Figure 2. Note that the performance of the machines for a small DSMC benchmark problem varied by a factor of almost 40, and the available memory varied by over a factor of four.

Processor	Memory (MB)	Operating System	Relative Speed
30 MHz Sparc	128	SunOS 4.1.3	1.0
150 MHz Alpha	64	Digital UNIX 3.2	4.4
133 MHz R4600	64	IRIX 5.3	6.0
133 MHz R4600	64	IRIX 6.2	6.0
133 MHz R4600	64	IRIX 6.2	6.0
150 MHz R4400	288	IRIX 6.2	6.8
200 MHz R4400	128	IRIX 5.3	8.6
200 MHz Pentium	64	Windows NT 4.0	13.0
180 MHz R10000 (x2)	128	IRIX 6.4	38.0
266 MHz Pentium II (x2)	256	Windows NT 4.0	39.0

Figure 2: Processor architectures, memory capacities, operating systems and relative speeds of computers in heterogeneous testbed.

Four experiments were conducted. First, the problem was run without load balancing. In this case, time steps required an average of 14.1 seconds each. Next, a homogeneous load balancing strategy was used with CPU time as the task loads. Although CPU time is actually a utilization metric and not a load metric, it can be used as the load if the computers are considered to be homogeneous. If a capacity-invariant quantity such as the particle count were used, no tasks would have been moved, since all of the computers initially had the same number of particles. As in all the remaining cases, load balance was achieved by dynamically repartitioning the problem and redistributing the resulting partitions according to the methods described above. After homogeneous load balancing, simulation steps required 6.5 seconds each. Also, in subsequent load balancing steps, computers continued to transfer large numbers of tasks, without improving the step time. This was due to the absence of computer capacity estimates; the utilizations of the computers did not vary as the load balancing algorithms expected. For example, transferring 10 seconds of work from one computer to another might change the utilization of the latter computer by much more or much less that 10 seconds. In the third test case, a small benchmark problem, roughly 20% as large as the full problem, was run on each machine. Using these static capacity estimates, the problem was balanced. The time per step dropped to 2.5 seconds. Unlike the homogeneous case, the number of tasks transferred dropped off rapidly after the first two load balancing rounds. A few tasks continued to be transferred in subsequent load balancing rounds, however, due to the differences between the capacity estimates and the actual capacities of the computers. In the final test, the computers' capacities were calculated dynamically by dividing the total number of particles on each computer by the CPU time required to process them. This improved performance even more, reducing the step time to 2.0 seconds. No further task transfers took place after the third load balancing round, as the capacity estimates were quite exact. These results are summarized in Figure 3. Because the improvement numbers in Figure 3 were skewed by the presence of a very slow computer (the SparcServer), which made the unbalanced case extremely slow, another round of tests were conducted in which that machine was omitted. The results from those tests are given in Figure 4.

Scenario	Step Time (sec)	Improvement
No LB	14.1	None
Homogeneous LB	6.5	2.2x
Heterogeneous LB with static capacities	2.5	5.6x
Heterogeneous with dynamic capacities	2.0	7.1x

Figure 3: Results of load balancing experiments for entire heterogeneous testbed.

Scenario	Step Time (sec)	Improvement
No LB	4.3	None
Homogeneous LB	5.2	None
Heterogeneous LB with static capacities	2.4	1.8x
Heterogeneous LB with dynamic capacities	2.0	2.2x

Figure 4: Results of load balancing experiments for heterogeneous testbed without the slowest machine.

CONCLUSION

As this work has shown, substantial improvements in performance are possible when one takes into account the individual resource capacities of the computers on which a concurrent application is running. The extensions required to existing load balancing frameworks are fairly simple ones. These modifications can be combined with our previous work on vector load balancing techniques (Watts *et al.* 1997) to simultaneously redistribute the utilization of multiple resources, such as both processing time and memory. Coupled together, these techniques allow the effective use of machines with disparate memory and computational capabilities.

ACKNOWLEDGEMENTS

This research was sponsored by the Advanced Research Projects Agency under contract number DABT63-95-C-0116. Equipment in the heterogeneous testbed was purchased with support from the Ballistic Missile Defense Organization under project number DAAH04-96-1-0319 and the National Science Foundation under project number AFS-9157650.

BIBLIOGRAPHY

Boillat, J. 1990. "Load Balancing and the Poisson Equation in a Graph." *Concurrency: Practice and Experience*, vol. 6: 101-117.

Bowen, N.; C. Nikolaou; and A. Ghafoor. 1992. "On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems." *IEEE Transactions on Computers*, vol: 41: 257-273.

Cap, C. and V. Strumpen. 1993. "Efficient Parallel Computing in Distributed Workstation Environments." *Parallel Computing*, vol. 19: 1221-1234.

Cybenko, G. 1989. "Dynamic Load Balancing for Distributed Memory Multiprocessors." *Journal of Parallel and Distributed Computing*, vol. 7: 279-301.

Heirich, A. and S. Taylor. 1995. "A Parabolic Load Balancing Algorithm." *Proceedings of the 24th International Conference on Parallel Programming*, vol. 3. CRC Press, 192-202.

Henriksen, P. and R. Keunings. 1994. "Parallel Computation of the Flow of Intergral Viscoelastic Fluids on a Heterogeneous Network of Workstations." *International Journal for Numerical Methods in Fluids*, vol: 18: 1167-1183.

Li, K. and J. Dorband. 1997. "A Task Scheduling Algorithm for Heterogeneous Processing." Proceedings of *High Performance Computing '97*, SCS, 183-188.

Lin, H.-C. and C. Raghavendra. 1992. "A Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC)." *IEEE Transactions on Software Engineering*, vol. 18: 148-158.

Rieffel, M.; S. Taylor; J. Watts; and S. Shankar. 1997. "Concurrent Simulation of Plasma Reactors." Proceedings of *High Performance Computing '97*, SCS, 163-168.

Taylor, S.; J. Watts; M. Rieffel; and M. Palmer. 1996. "The Concurrent Graph: Basic Technology for Irregular Problems." *IEEE Parallel and Distributed Technology*, vol. 4:979-993.

Watts, J.; M. Rieffel; and S. Taylor. 1997. "A Load Balancing Technique for Multiphase Computations." Proceedings of *High Performance Computing '97*, SCS, 15-20.

Watts, J. and S. Taylor. 1998. "Communication Locality Preservation in Dynamic Load Balancing." To appear elsewhere in these proceedings.

Watts, J. and S. Taylor. 1998. "A Practical Approach to Dynamic Load Balancing," to appear in *IEEE Transactions on Parallel and Distributed Computing.*

Willebeek-LeMair, M. and A. Reeves. 1993. "Strategies for Dynamic Load Balancing on Highly Parallel Computers." *IEEE Transactions on Parallel and Distributed Systems*, vol. 4:979-993.

Xu, C. and F. Lau. 1997. *Load Balancing in Parallel Computers: Theory and Practice.* Kluwer Academic Publishers, Boston, Mass.

Zhou, S.; X. Zheng; J. Wang; and P. Delisle. 1993. "Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems," *Software: Practice and Experience*, vol. 23: 1305-1336.