EFFICIENT RESOURCE MANAGEMENT FOR SCIENTIFIC APPLICATIONS IN DISTRIBUTED COMPUTING ENVIRONMENT

IOANA BANICESCU, SHEIKH K. GHAFOOR and
MARK BILDERBACK
Mississippi State University
Department of Computer Science and
NSF Engineering Research Center for Computational Field Simulation
E-mail: {ioana, ghafoor, bilderba}@cs.msstate.edu

Abstract
Resource management for scientific applications in a vast distributed computing environment, such as the world-wide interconnected networks (Internet, Web), is a complex problem. This paper investigates various techniques used to manage resources in a network of workstations at both coarse and fine levels of granularity. A new strategy which combines two competitive techniques, one from each granularity level, is proposed. Further, a discussion of how this strategy can be extended to the world-wide interconnected networks, is presented.

Keywords: Distributed Computing, Resource Management, Web, Network of Workstations, Load Balancing, Hectiling

1. INTRODUCTION

Scientific applications are very large and require extensive computational resources. Traditionally, most of these applications were executed on massively parallel computers. With the increase of the computational power of desktop workstations and advances in the network technology, network of workstations (NOW) has become a widespread platform for parallel computing. Applications from domains such as computational field simulation have successfully exploited the NOW environment. The advantages of such an environment are low initial hardware and maintenance costs, as well as prevalence and availability versus high cost and scarcity of supercomputers. One of the disadvantages of such an environment is high communication cost. Another, is limited control over load imbalance, resulting in performance degradation and poor resource utilization.

With the recent expansion of the world-wide interconnected networks, (which can be considered a vast NOW) the execution of scientific applications on remotely connected networks has become more feasible. This will offer the user the flexibility to take advantage of newly developed state-of-the-art tools, techniques, and applications, as soon as they become available, as opposed to waiting for them to become widely distributed and compatible with the environment of choice. In addition, this will allow the user to have a greater selection of tools, techniques, and applications with minimal investment.

This paper focuses on the efficient utilization of resources for scientific applications in a distributed computing environment and considers the possibility of extending this approach to the world-wide interconnected networks. It concentrates on improving the performance of scientific applications for better resource utilization through load balancing.

Coarse grain load balancing is achieved by moving tasks from heavily to lightly loaded nodes (In this paper nodes, processors and workstations are used interchangeably). On the other hand, fine grain load balancing is usually achieved by data migration. Both techniques have their advantages and disadvantages. By combining them, the new system could offer the advantages of both, resulting in better resource utilization. *Hector*, a parallel runtime environment developed at Mississippi State University performs coarse grain load balancing by migrating tasks [17]. *Fractiling* is a fine grain technique that achieves load balancing by migrating data from heavily loaded to idle tasks [2]. It has been successfully applied to N-body simulations. In this paper, we discuss the combination of these two strategies, Hector and fractiling, to achieve better load balancing. The combined system, resulting in better resource utilization, is proposed to be extended to the world-wide interconnected networks.

Sections §2, §3, and §4, describe Hector, fractiling, and the combination of these two methods, respectively. Section §5 presents conclusions and considerations for implementing the proposed system on the world-wide interconnected networks.

2. COURSE GRAIN RESOURCE MANAGEMENT

In a distributed computing environment, such as a network of workstations, performance of scientific applications could be significantly improved by balancing the work load. A certain degree of load balancing can be achieved by moving tasks from highly loaded to lightly loaded or idle nodes. Various parallel programming models and environments have been proposed to achieve such coarse grain load balancing.

One such system, named CARMI, allocates Parallel Virtual Machine (PVM) tasks across idle workstations and migrates PVM tasks as workstations fail or become heavily loaded [14]. The disadvantages of CARMI are its inability to claim newly available resources and the increased bottleneck due to checkpointing of all tasks even when only one task needs to be migrated. Another similar system is the Prospero Resource Manager (PRM)[13]. All programs that run under PRM have a job manager which negotiates the acquisition of system resources. PRM is scheduled to use elements of Condor to support task migration and checkpointing. It requires a custom written job manager for each application. A recently proposed system, the Distributed Queuing System (DQS), is designed to manage jobs simultaneously across multiple computers [11]. It can support one or more queue masters, which process user requests for resources on a first-come-first-serve (FCFS) basis. Resource allocation is progressively performed as applications start. The DQS has no support for task migration or fault tolerance. It supports both

PVM and Message Passing Interface (MPI) applications. There are several other operating environments which provide load balancing by task migration. Research efforts in managing NOW (as the ones presented above) and a list of commercial products with their characteristics, are surveyed in detail in [1].

Hector [17], a parallel runtime operating environment developed at Mississippi State University, provides dynamic load balancing by task migration. It supports MPI applications and is currently implemented for the Sun and SGI workstations. Hector attempts to balances the load by migrating task(s) during execution, from heavily loaded to idle or lightly loaded workstations. It has the facility to create backup copies of running programs, thereby providing fault tolerance. Hector is designed to use a master-slave hierarchy. A single task, called "master allocator", runs on one workstation performing all decision making functions. Instead of directly controlling MPI programs, the master allocator communicates with tasks called "slave allocators". There is only one slave allocator for each physical machine. Each slave allocator launches and controls all MPI tasks running on its machine. In addition, it monitors performance characteristics of the MPI tasks. The slave allocator periodically reports performance information to the master allocator. The master allocator then makes decisions regarding allocation and migration. When a decision has been made to migrate a task, the master allocator sends migration commands to the slave allocator which had previously started the task. For a more detailed description of Hector and its functionality refer to [15][17][19]. Comparisons between Hector features and those of similar systems can be found in [18].

The Hector runtime environment ensures dynamic load balancing at a coarser granularity level. In the next section a dynamic scheduling technique that provides load balancing at a finer granularity level is presented.

3. RESOURCE MANAGEMENT FOR DATA-PARALLEL APPLICATIONS

Many algorithms used in solving irregular and computationally intensive scientific problems are amenable to parallel execution. Performance gains from parallel execution are difficult to obtain, due to load imbalance. Imbalance could be caused by irregularity of data distribution, as well as by different processing requirements of data in interior versus near computation space boundary. In addition, the data distribution may vary at each time step.

Problems in scientific computing have previously employed various methods to balance processor loads and to exploit locality. For unstructured problems, static partitioning and repetitive static partitioning heuristics have been the only methodology used so far to overcome dynamic load imbalance. Most of these methods use profiling by gathering information on the work load from a previous time step in the execution of the algorithm, in order to estimate the optimal work load distribution at the present time step. The cost of these methods increases with the number of processors and problem size [21][24]. Moreover, these methods employ a static assignment of work load to processors during a time step, due to an assumption that the distribution of particles changes slowly between time steps. These assumptions are not valid in the entire spectrum of scientific applications and therefore these methods are not robust.

In a large number of scientific problems, domain decomposition methods for optimal load balancing need considerable improvements [14]. This is especially true in the case of applications where none of the existing load balancing strategies accommodate the unpredictable behavior of these simulations (i.e. plastic deformations, nonisothermal multiphase flow, thermomechanical fatigue, modeling of radiation and magnetohydrodynamics, shock physics analysis, explosions, etc.).

Processor load imbalances are induced not only by application features, such as irregular data and conditional statements, but also by system effects, such as data access latency and operating system interference. Adapting to system induced load imbalances requires dynamic work assignment. Dynamic scheduling schemes attempt to maintain balanced loads by assigning work to idle processors at runtime. Thus, they accommodate systemic as well as algorithmic variances. In general, there is a tension between exploiting data locality and dynamic load balancing as the re-assignment of work may necessitate access to remote data. The cost of dynamic schemes is loss of locality, which translates in increased overhead.

Load balancing techniques have been extensively applied to N-body simulations by using information about data (particles) distribution to guide the static assignment of data to processors [22][24][20][6][7]. The assignment is recomputed after each time step as data changes over time. Some of these techniques include the orthogonal recursive bisection (ORB), and the Costzones methods [23][22]. Others use a hash function to build the hashed oct-tree (HOT) which employs Morton order, a space-filling numbering scheme [24]. Randomly assigning subtiles of a certain size to processors has also been considered to improve the performance of N-body simulations due to load imbalance [8]. With random assignment, the load imbalances of individual subtiles mute each other out to some extent.

Some experimentation with new scheduling schemes applied to scientific problems have been presented in [10][2][5] [12]. These schemes combine static techniques that exploit data locality with dynamic techniques that improve load balancing. In these schemes, work units and their associated data are initially placed on the same processor. Each processor executes its units in decreasing size chunks to preserve load balancing. After exhausting its local work, each processor acquires decreasing size chunks of work from other processors. These decreasing chunks are represented by multidimensional subtiles of the same shape selected to maximize data reuse. The subtiles are combined in Morton order in larger subtiles, thus preserving the self-similarity property [4][5]. In this way, a complex history of executed subtiles does not need to be maintained.

Fractiling, a dynamic scheduling technique that incorporates the above features, balances processor loads and maintains locality by exploiting self-similarity properties of fractals [10][3]. This technique has been applied to N-body simulations using the parallel implementation of the Greengard's 3-d Fast Multipole Algorithm [9] on both a distributed memory shared-address space and a message passing environment. The distributed memory shared-address space implementation was run on a KSR-1 at the Cornell Theory Center [2][4] and the message passing environment implementation was run on an IBM SP2 at the Maui High Performance Computing Center [5][12].

Fractiling is based on a probabilistic analysis. It thus accommodates load imbalances caused by predictable events (such as irregular data) and unpredictable events (such as data access latency). Fractiling adapts to algorithmic and system induced load imbalances while maximizing data locality. In fractiling, work and the corresponding data are initially placed to processors in tiles, to maximize locality. The processors that finish early "borrow" decreasing size subtiles of work units from slower processors to balance loads. The sizes of these subtiles are chosen so that they have a high probability of finishing before the optimal time. The subtile assignment are computed in an efficient way by exploiting the self-similarity property of fractals.

The approach used to load balance N-body simulations in this environment was to incorporate fractiling into the N-body code that discretizes the space into an oct-tree. Implementations of a parallel and a fractiled N-body simulation on uniform and nonuniform distributions of particles of various sizes and using up to 64 processors were compared. Experimental work in both uniform and nonuniform distributions of particles confirmed that fractiled N-body simulations consistently resulted in improved performance[2][12].

The next section presents an integration of fractiling into a task parallel load balancing strategy to improve the overall resource utilization in a distributed computing environment.

4. COMBINED STRATEGY FOR EFFICIENT RESOURCE UTILIZATION

A Hector-like coarse grain load balancing system can achieve better resource utilization by migrating task from highly loaded workstations to idle or lightly loaded workstations in a distributed computing environment. Since the sizes of tasks are unequal, this coarse grain load balancing strategy continues to suffer from load imbalance. This results in poor resource utilization. On the other hand, a fine grain data parallel load balancing strategy, such as fractiling will ensure a high degree of load balancing by migrating data from one workstation to another Since fractiling does not support task migration, a fractiled scientific application may suffer from load imbalance and poor resource utilization in a distributed computing environment. Consider a scenario in which a fractiled scientific application is running in a distributed computing environment. Suddenly, while fractiled tasks are running, one or more workstations become overloaded due to additional external load. The fractiling algorithm will now balance the load by migrating data from tasks running on overloaded workstations to lightly loaded workstations. Suppose some workstation are idle; if fractiling had the capability of task migration in a Hector-like fashion, the fractiled tasks from the overloaded workstations could have been migrated to idle workstations and, in this way, better resource utilization would have been achieved. Hence, combining the benefits of both fractiling and Hector into a single system would have resulted in better resource utilization.

To take advantage of the benefits offered by Hector and fractiling, a new system integrating both is proposed. Fractiling-like data parallel applications require communication to control the exchange of data between tasks. This process requires knowledge of both busy and idle tasks. Detection of idle tasks is simple, since idle tasks can always send requests for additional work. However, the detection of busy tasks is more difficult, since it requires ongoing knowledge of the status of all tasks. The Hector environment is equipped to provide this knowledge through its information gathering infrastructure. Likewise, requests for additional work from idle workstations, can be channeled through Hector.

An architecture for running a fractiling application under Hector, Hectiling, is described in [16]. The slave allocators continuously monitor the tasks and their nearness to completion. For example, nearness to completion can be monitored by checking local iteration loop counters. This information is forwarded to the master allocator as part of the periodic information gathering process. Once they complete their current work assignment, individual tasks ask the fractile master for additional work through the Hector master allocator. The master allocator forwards the state of all tasks and the explicit work load requests to the fractiling master The fractiling master can use each task's status to estimate its nearness to completion. Work load can then be transferred from relatively "busy" tasks to idle ones. Since the state of each task is being monitored by the slaves, busy tasks can be identified. Since idle tasks post explicit requests for additional work load, they are already known.

For testing purposes, two implementations of N-body simulations using the Parallel Fast Multipole Algorithm, one with and one without fractiling, have been used [12]. These implementations written in C and using MPI were run on datasets of 100k particles. These experiments were conducted on a cluster of 8 quad-processor, 90 MHz SPARCstation 10's. Three different data distributions were used: a uniform distribution of particles ("Uniform"), a Gaussian distribution of particles centered on the grid space ("Gaussian"), and a Gaussian distribution of particles shifted towards one corner of the computation space ("Corner"). The fractiled and non-fractiled cases were run with Hector on 4 to 32 processors. During these executions the system was exclusively used for these experiments. As a measure of resource utilization, the coefficients of variation (C.O.V.) of processor finishing times have been computed. Results for all distributions are shown in table 1. These results show that Hectiling significantly improves resource utilization.

5. CONCLUSIONS

There are several advantages of running self-balancing fractiling applications under a task parallel load-balancing runtime system. First, self-balancing applications allow the rapid release of busy resources and reclamation of idle ones. Second, construction of data-parallel applications is simplified because the runtime system already has the capability of information-gathering, therefore relieving the self-balancing application from this responsibility. In addition, the combined system has more knowledge about the system, because the runtime system gathers more detailed information about the performance at individual nodes. The knowledge acquired through the information gathering may facilitate the possibility of data prefetching. Furthermore, scientific applications with different work load profiles may be executed simultaneously and scheduled cooperatively.

Particle Distrib.	# of Processors	PFMA c.o.v.	Fractiling c.o.v.	% Improvement
Uniform	4	0.00719	0.00025	96.00
	8	0.00886	0.00058	93.28
	16	0.11009	0.00972	91.17
	32	0.13298	0.03197	75.96
Gaussian	4	0.01218	0.00095	92.23
	8	0.01860	0.00365	80.89
	16	0.65480	0.24522	62.55
	32	0.93900	0.42940	54.27
Corner	4	0.56758	0.23270	59.00
	8	0.66997	0.14415	78.48
	16	0.66776	0.12640	81.07
	32	0.68977	0.10402	84.82

Table 1: Improvements in c.o.v. due to Hectiling

The world wide web can be thought of as a vast loosely coupled network of workstations. The idea of Hectiling can be extended to the world wide web. This would allow the resources of the web to be effectively used for running scientific applications. However, our proposal for a web operating system would require the built-in Hectiling-like features described above and features which would address, among others, the following general concerns.

1. Where will the master allocator reside? A possible solution is that the master will reside on any machine which the user has access to. The user will launch the master on his machine which in turn will then acquire the slaves and start the application. Another possibility would be to have regional master allocators. All applications in a region will be submitted to the regional master who will then acquire the slaves and launch the application.

2. How will the slave allocators be started? One possibility is that all participating machines have a slave allocator process running in a daemon-like fashion.

3. How many masters allocators can a slave allocator serve? The number could be one master per slave, in a first-come-first-serve manner. Alternately, multiple masters per slave could be considered.

4. How will the master acquire the slaves? This acquisition needs to be transparent to the user and slaves selected must be compatible with the application (i.e. operating system and hardware).

5. How will a master and a slave communicate? A uniform widely accepted protocol has to be developed for the communication between masters and slaves.

While these are only a few issues of concern, this is a new area of research and therefore more issues will be identified as the work continues in this field.

Acknowledgments

We greatly appreciate Dr. Samuel Russ and Rong Lu for their invaluable input regarding Hector and the recent implementations of the N-body simulation.

REFERENCES

[1] M. Baker, G. Fox and H. Yau: "*Cluster Computing Review*".-
Northeast Parallel Architecture Center, Syracuse University, www.npac.syr.edu/techreports/hypertext/sccs-0748/cluster-review.htm, (1995)

[2] I. Banicescu: "*Load Balancing and Data Locality in the Parallelization of the Fast Multipole Algorithm*".-
PhD thesis, Polytechnic University, (1996)

[3] I. Banicescu and S. F. Hummel: "*Balancing Processor Loads and Exploiting Data Locality in Irregular Computations*".-
Technical Report, IBM, RC19934, (1995)

[4] I. Banicescu and S. F. Hummel: "*Balancing Processor Loads and Exploiting Data Locality in N-Body Simulations*".-
in: Proceedings of Supercomputing'95 Conference, (1995)

[5] I. Banicescu and R. Lu: "*Experiences with Fractiling in N-Body Simulations*".-
to appear in: Proceedings of High Performance Computing'98 Symposium, (1998)

[6] J. A. Board, J. Causey, J. F. Leathrum Jr. and others: "*Accelerated Molecular Dynamic Simulations with the Parallel Fast Multipole Algorithm*".-
in: Chemical Physics Letters, 23-34, (1992)

[7] J. A. Board, Z. S. Hakura, W. D. Elliot and others: "*Scalable Variants of Multipole-based Algorithms for Molecular Dynamics Applications*".-
in: Proceeding of Seventh SIAM Conference on Parallel Processing for Scientific Computing, 295-300, (1995)

[8] A. Y. Grama, V. Kumar and A. Sameh: "*Scalable Parallel Formulations of Barnes-Hut Method for N-Body Simulations*".-
in: Proc. of Supercomputing'94,439-448, (1994)

[9] L. Greengard and Vladimir Rokhlin: "*A fast algorithm for particle simulation*".-
in: Journal of Computational Physics, 325-348, (1987)

[10] S. F. Hummel and E. Schonberg and L. E. Flynn: "*A Practical and Robust Method for Scheduling Parallel Loops*".-
in: Communications of the ACM, 90-101, (1992)

[11] Supercomputing Research Institute: "*DQS User Manual - DQS Version 3.1.2.3*".-
Florida State University, (1995)

[12] R. Lu: "*Parallelization of the Fast Multipole Algorithm with Fractiling in Distributed Memory Architectures*".-
Master's thesis, Mississippi State University, (1997)

[13] B. Neuman and S. Rao: "*The Prospero Resource Manager: A Scalable Framework for Processor Allocation in Distributed System*".-
in: Concurrency: Practice and Experience, 339-355, (1994)

[14] J. Pruyne and M. Livney: "*Providing Resource Management Services to Parallel Applications*".-
in: Workshop on Job Scheduling Strategies for Parallel Processing, Proceedings of the International Parallel Processing Symposium (IPPS 1995, (1995)

[15] J. Robinson, S. Russ, B. Flachs and B. Heckel: "*A Task Migration Implementation of the Message Passing Interface*".-
in: 5th High Performance Distributed Computing Conference (HPDC-5), (1996)

[16] S. Russ, I. Banicescu, S. Ghafoor, B. Janapareddi, J. Robinson and R. Lu: "*Hectiling: An Integration of Fine and Coarse-Grained Load-Balancing Strategies*".-
Submitted to: International Symposium on High Performance Distributed Computing '98", (1998)

[17] S. Russ, B. Flachs, J. Robinson and B. Heckel: "*Hector: Automated Task Allocation for MPI*".-
in: 10th International Parallel Processing Symposium, (1996)

[18] S. Russ, M. Gleeson, B. Meyers, L. Rajagopalan and C. Tan: "*Using Hector to RUN MPI Programs over Networked Workstation*".-
to appear in: Concurrency: Practice and Experience, (1998)

[19] S. Russ, B. Meyers, M. Gleeson, J. Robinson, L. Rajagopalan, C. Tan and B. Heckel: "*User Transparent Run-Time Performance Optimization*".-
in: The 2nd International Workshop on Embedded HPC and Applications at the 11th IEEE International Parallel Processing Symposium, (1997)

[20] J. Salmon and M. S. Warren: "*Parallel, Out-of-core methods for N-body Simulation*".-
in: Proceeding of 8th SIAM Conference on Parallel Processing for Scientific

Computing, (1997)

[21] J. Singh: "*Parallel Hierarchical N-body Methods and their Implications for Multiprocessors*".-
PhD Thesis, Stanford University, (1993)

[22] J. Singh, C. Holt, T. Totsuka and others: "*A Parallel Adaptive Fast Multipole Algorithm*".-
in: Proc. of Supercomputing'93, 54-65, (1993)

[23] M. Warren and J. Salmon: "*Astrophysical N-Body Simulation using Hierarchical Tree Structures*".-
in: Proc. of Supercomputing'92, (1992)

[24] M. Warren and J. Salmon: "*A Parallel Hashed Oct Tree N-body Algorithm*".-
in: Proceeding of Supercomputing'93, 12-21, (1993)

Authors:

Ioana Banicescu is an Assistant Professor in the Department of Computer Science at Mississippi State University, with research involvement at the National Science Foundation Engineering Research Center for Computational Field Simulation. Her research interests include parallel algorithms, scientific computing, scheduling theory and load balancing algorithms.

Sheikh K. Ghafoor is a Master's candidate in the Department of Computer Science at Mississippi State University and a Research Assistant at the National Science Foundation Engineering Research Center for Computational Field Simulation. His research interests include run-time environments for networks of workstations and load balancing algorithms.

Mark Bilderback is a Ph.D. candidate in the Department of Computer Science at Mississippi State University and a Research Assistant at the National Science Foundation Engineering Research Center for Computational Field Simulation. His research interests include scheduling theory, load balancing algorithms and performance analysis.