Performance-Based Workload Distribution on Grid Environments*

Wen-Chung Shih[1], Chao-Tung Yang[2,**], Tsui-Ting Chen[2], and Shian-Shyong Tseng[1,3]

[1] Department of Computer Science
National Chiao Tung University, Hsinchu, 30010, Taiwan (R.O.C.)
{gis90805, sstseng}@cis.nctu.edu.tw
[2] High-Performance Computing Laboratory
Department of Computer Science and Information Engineering
Tunghai University, Taichung, 40704, Taiwan (R.O.C.)
{ctyang, g95280003}@thu.edu.tw
[3] Department of Information Science and Applications
Asia University, Taichung, 41354, Taiwan (R.O.C.)
sstseng@asia.edu.tw

Abstract. Imbalanced workload-distribution can significantly degrade performance of grid computing environments. In the past, the theory of divisible load has been widely investigated in static heterogeneous systems. However, it has not been widely applied to grid environments, which are characterized by heterogeneous resources and dynamic environments. In this paper, we propose a performance-based approach to workload distribution for master-slave types of applications on grids. Furthermore, applications with irregular workloads are addressed. We implemented three kinds of applications and conducted experimentations on our grid test-beds. Experimental results show that this approach performs more efficiently than conventional schemes. Consequently, we claim that dynamic workload distribution can benefit applications on grid environments.

1 Introduction

Grid platforms, which consist of various computational and storage resources, have become promising alternatives to traditional multiprocessors and computing clusters [3, 4, 7-9, 14, 25-28, 40]. The goal of grid computing is to share resources through the internet. Therefore, users can access more computing resources through grid technologies. On the other hand, inappropriate management of grid environments might result in using grid resources in an inefficient way. Moreover, the characteristic of dynamic changing makes it different from conventional parallel and distributed computing systems, such as multiprocessors and computing clusters. Consequently, it is challenging to use the grid efficiently.

* This work was partially supported by National Science Council of Republic of China under the number of NSC95-2752-E-009-015-PAE.

** Corresponding author.

C. Cérin and K.-C. Li (Eds.): GPC 2007, LNCS 4459, pp. 385 – 396, 2007.

In the past, the master-slave paradigm is a common model for task dispatching in parallel and distributed computing environments [16]. In this model, the master node holds a pool of tasks to be dispatched to other slave nodes. A well-known application of this model is Divisible Load Theory (DLT) [1, 17-19, 32, 36], which deals with the case where the total workload can be partitioned into any number of independent subjobs. In [23], a data distribution method was proposed for host-client type of applications. Their method was an analytic technique, and only verified on homogeneous and heterogeneous cluster computing platforms. In [24], an exact method for divisible load was proposed, which was not from a dynamic and pragmatic viewpoint as ours.

This paper aims to address the problem of dynamic distribution of workload for master-slave applications on grids. Since grid environments are dynamically changing and heterogeneous, the problem is more challenging than the traditional DLT problem. We propose a performance-based approach, which is implemented in three types of applications, Matrix Multiplication, Association Rule Mining and Mandelbrot Set Computation, and is executed a grid test-bed. Experimental results show that effective workload partitioning can significantly reduce the total completion time.

Our major contributions can be summarized as follows. First, this paper proposes a new performance function to estimate the performance of grid nodes. Second, we apply this approach to programs with irregular workload distribution. Consequently, experimental results show the obvious effectiveness of our approach. Our previous work [37-39] presents different heuristics to the parallel loop self-scheduling problem. This paper generalizes their main idea and proposes to solve the dynamic workload distribution problem. This approach is applied to both the parallel loop self-scheduling application and the association rule mining application. There have been a lot of researches of parallel and distributed data mining [12, 13, 29, 47]. However, this paper focuses on workload distribution, instead of proposing a new data mining algorithm.

The remainder of this paper is organized as follows. In Section 2, background on parallel loop scheduling and association rule mining is reviewed. In Section 3, we describe the proposed approach to solve the dynamic workload distribution problem. Next, the configuration of our grid testbed is specified and experimental results on three types of applications are also presented in Section 4. Finally, the concluding remarks are given in the last section.

2 Background Review

In this section, parallel loop scheduling and association rule mining are briefly reviewed.

2.1 Dynamic Loop Scheduling Schemes

Dynamic loop scheduling schemes make a scheduling decision at runtime. Its disadvantage is more overhead at runtime, while the advantage is load balance. The schemes we focus in this paper are self-scheduling, which a large class of dynamic loop scheduling schemes. Several self-scheduling schemes have been reviewed in [15, 21, 22, 30, 33, 41, 42, 46], and they are restated here as follows.

- **Pure Self-scheduling (PSS).** This is a straightforward dynamic loop scheduling algorithm [32]. Whenever a processor becomes idle, a loop iteration is assigned to it. This algorithm achieves good load balance but also induces excessive overhead.

- **Chunk Self-scheduling (CSS).** Instead of assigning one iteration to an idle processor at one time, CSS assigns k iterations each time, where k, called the chunk size, is a constant.
- **Guided Self-scheduling (GSS).** This scheme can dynamically change the number of iterations assigned to each processor [35]. More specifically, the next chunk size is determined by dividing the number of remaining iterations of a parallel loop by the number of available processors.
- **Factoring Self-scheduling (FSS).** The Factoring algorithm addresses this problem [31]. The assignment of loop iterations to working processors proceeds in phases. During each phase, only a subset of the remaining loop iterations (usually half) is divided equally among the available processors.
- **Trapezoid Self-scheduling (TSS).** This approach tries to reduce the need for synchronization while still maintaining a reasonable load balance [43]. This algorithm allocates large chunks of iterations to the first few processors and successively smaller chunks to the last few processors.

In [44], the authors enhanced well-known loop self-scheduling schemes to fit an extremely heterogeneous PC cluster environment. A two-phased approach was proposed to partition loop iterations and it achieved good performance in heterogeneous test-beds. In [20, 45, 46], NGSS was further enhanced by dynamically adjusting the parameter α according to system heterogeneity. A performance benchmark was used to determine whether target systems are relatively homogeneous or relatively heterogeneous. In addition, the types of loop iterations were classified into four classes, and were analyzed respectively. The scheme enhanced from GSS is called ANGSS in this paper.

2.2 Association Rule Mining

The objective of association rule mining is to discover correlation relationships among a set of items [29]. The well-known application of association rule mining is market basket analysis. This technique can extract customer buying behaviors by discover what items they buy together. The managers of shops can place the associated items at the neighboring shelf to raise their probability of purchasing. For example, milk and bread are frequently bought together.

The formulation of association rule mining problem is described as follows [12-13]. Let $I=\{I_1, I_2, I_3, \ldots, I_m\}$ be a set of items, and D a database of transactions. Each transaction in D is a subset of I. An association rule is a rule of the form $A \Rightarrow B$, where $A \subset I$, $B \subset I$, and $A \cap B = \{\}$. The well-known algorithm for finding association rules in large transaction databases is Apriori. It utilizes the Apriori property to reduce the search space.

As the rising of parallel processing, parallel data mining have been well investigated in the past decade. Especially, much attention has been directed to parallel association rule mining. A good survey can be found in [47].

3 Approach: Performance-Based Workload Distribution (PWD)

In this section, the system and programming model is introduces first. Then, the parameters of performance ratio and static-workload ratio are described. Finally, we present the skeleton algorithm for the performance-based workload distribution.

3.1 The System Model

The system in this work is modeled by a master-slave paradigm, which is represented by a star graph, G = (N, E). In this graph, N means the set of all nodes on the grid, and E is the set of all edges between the master and the slaves. In this model, there are two kinds of attributes associated with nodes, constants and variables. The values of the constant attributes do not vary during the lifetime of the node. For example, CPU clock speed, memory size, etc. are all constant attributes. On the other hand, the values of the variable attributes may fluctuate during the lifetime of the node. For example, CPU loading, available memory size, etc. are all constant attributes. In the following sections, the two kinds of attributes are utilized to model the heterogeneity of the dynamic grid.

3.2 Performance Ratio

The concept of performance ratio was previously defined in [37-39] in different forms and parameters, according to the requirements of applications. In this work, a different formulation is proposed to model the heterogeneity of the dynamic grid nodes. The purpose of calculating performance ratio is to estimate the current capability of processing for each node. With this metric, we can distribute appropriate workloads to each node, and load balancing can be achieved. The more accurate the estimation is, the better the load balance is.

To estimate the performance of each slave node, we define a performance function (PF) for a slave node *j* as

$$PF_j(V_1, V_2, \ldots, V_m) \tag{1}$$

where V_i, $1< i <m$, is a variable of the performance function. In more detail, the variables could include CPU speed, networking bandwidth, memory size, etc. We propose to utilize a Grid Resource Monitoring Tool [11] to acquire the values of variable attributes for all slaves, and to acquire the values of constant attributes by MDS. In this paper, the *PF* for node *j* is defined as

$$PF_j = w_1 \times \frac{CS_j / CL_j}{\sum_{\forall node_i \in \mathrm{N}} CS_i / CL_i} + w_2 \times \frac{B_j}{\sum_{\forall node_i \in S} B_i} \tag{2}$$

where

- N is the set of all grid nodes.
- CS_i is the CPU clock speed of node i, and it is a constant attribute. The value of this parameter is acquired by the MDS service.
- CL_i is the CPU loading of node i, and it is a variable attribute. The value of this parameter is acquired by the Ganglia tool, as shown in Figure 1.
- B_i is the bandwidth (Mbps) between node i and the master node.
- w_1 is the weight of the first term.
- w_2 is the weight of the second term.

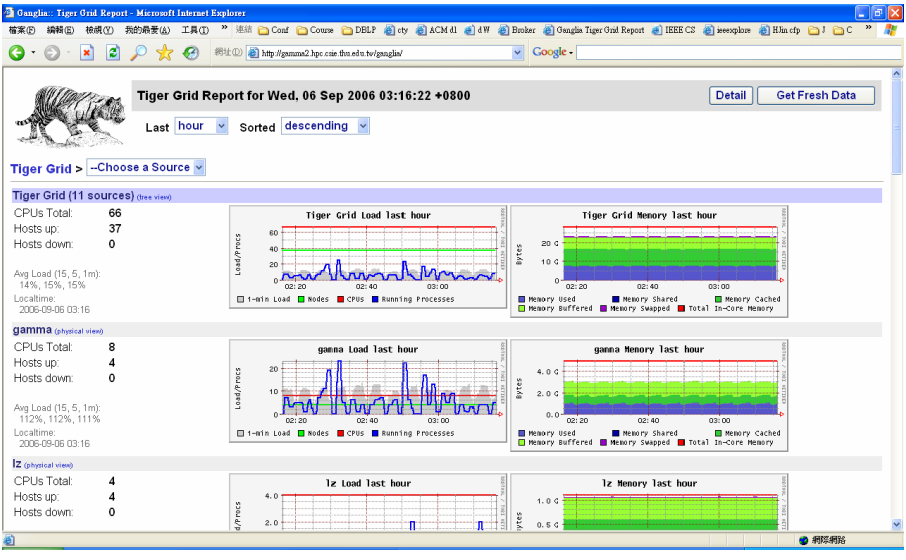

Fig. 1. The snapshot of the monitoring tool on the TIGER Grid

3.3 Determination of Static-Workload Ratio (*SWR*)

Another important factor to be estimated is the proportion of the workload which can be statically scheduled. For example, Mandelbrot Set Computation is a problem involving irregular workloads. In each iteration, the workload is different and varies significantly, as shown in Figure 2. Obviously, a distribution scheme which does not consider the effect of irregular workload could not estimate PR accurately.

We propose to use a parameter, *SWR* (Static-Workload Ratio), to alleviate the effect of irregular workload. In order to take advantage of static scheduling, *SWR* percentage of the total workload is dispatched according to Performance Ratio. If the workload of the target application is regular, *SWR* can be set to be 100. However, if the application has irregular workload, such as Mandelbrot Set Computation, it is reasonable to reserve some amount of workload for load balancing. We propose to randomly take five sampling iterations, and compute their execution time. Then, the *SWR* of the target application *i* is determined by the following formula.

$$SWR_i = \frac{min_i}{MAX_i} \tag{3}$$

where

- min_i is the minimum execution time of all sampled iterations for application *i*.
- MAX_i is the maximum execution time of all sampled iterations for application *i*.

For example, for a regular application with uniform workload distribution, the five sampled iterations are the same. Therefore, the *SWR* is 100%, and the whole workload can be dispatched according to Performance Ratio, with good load balance. However, for another application, the five sampling execution time might be 7, 7.5, 8, 8.5 and

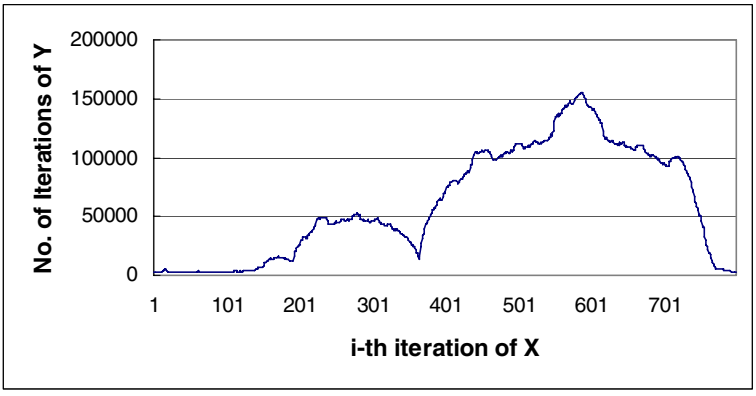

Fig. 2. The Mandelbrot Set on [-1.8, 0.5] to [-1.2, 1.2] an 800×800 pixel window

10 seconds, respectively. Then the *SWR* is 7/10, i.e. a percentage of 70. Therefore, 70 percentages of the workload would be scheduled statically according to *PR*, while 30 percentages of the workload would be scheduled dynamically by *GSS*.

3.4 Algorithm

Our algorithm is composed of four stages. In stage one, the related information are acquired. Then, stage two calculates the Static-workload Ratio and Performance Ratio. Next, *SWR* percentage of the total workload is statically scheduled according to the performance ratio among all slave nodes in stage three. Finally, the remainder of the workload is dynamically scheduled by Guided Self-Scheduling for load balancing. The algorithm of our approach is described as follows.

Module MASTER

```
Stage 1: Gathering the following information
       – CPU_Loading
       – CPU_Clock_Speed
       – the sample execution time
Stage 2: Calculate two scheduling parameters
Stage 3: Static Scheduling for SWR% of workload
Stage 4: dynamic Scheduling for the remaining
END MASTER
```

Module SLAVE

```
While (a chunk of workload arrives) {
   Receive the chunk of workload
   Compute on this chunk
   Send the result to the Master
}
END SLAVE
```

4 Experimental Results

To verify our approach, a grid test-bed is built based on the TIGER grid [11], and three types of application programs are implemented with MPI (Message Passing Interface) to be executed on this test-bed. This grid test-bed consists of one master and four domains, totally 33 nodes. The master node is at Tunghai University (THU), and the 32 slave nodes are located at Tunghai University (THU), Providence University (PU), Li-Zen High School (LZ), and Hsiuping Institute of Technology School (HIT). We have built this grid test-bed by the following middleware:

- Globus Toolkit 4.0.1 [2, 10]
- Mpich library 1.2.6 [5, 6]

In this study, we have implemented applications in C language, with message passing interface (MPI) directives for parallelizing code segments to be processed by multiple CPUs. For readability of experimental results, the brief description of all implemented programs is listed in Table 1.

Table 1. Description of all implemented programs

Scheduling Scheme	Description	Reference
static	Weighted static scheduling	
gss	Dynamic scheduling (GSS)	[35]
fss	Dynamic scheduling (FSS)	[31]
tss	Dynamic scheduling (TSS)	[43]
ngss	Fixed α scheduling + GSS	[44]
angss	Adaptive α scheduling + GSS	[46]
pwd	Performance-based Workload Distribution	

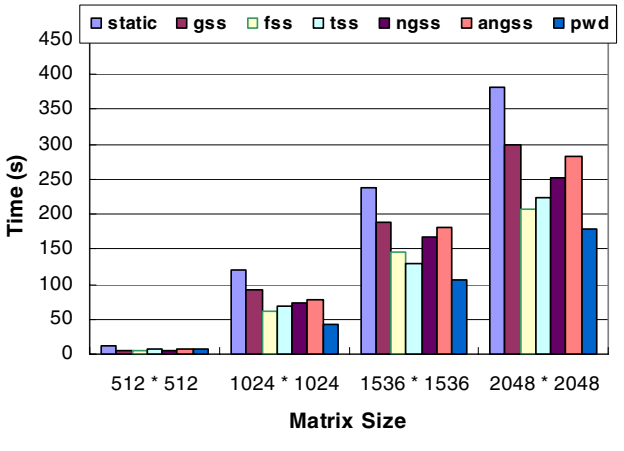

Fig. 3. Execution time for Matrix multiplication with different input sizes

4.1 Application 1: Matrix Multiplication

Matrix Multiplication is a fundamental operation in many numerical linear algebra applications. In this application, the workload is loop iterations. First, we want to compare the proposed PWD scheme with previous schemes with respect to the execution time. Figure 3 illustrates the execution time for input matrix size 512×512, 1024×1024, 1536×1536 and 2048×2048 respectively. The results are shown as follows.

- Among these schemes, PWD performs better than other schemes. The reason is that PWD accurately estimates the PR, and takes the advantage of static scheduling, thus reducing the runtime overhead.
- The weighted static scheme obviously performs worse than other dynamic schemes. It is reasonable to say that the static scheme is not suitable for a dynamic environment, with respect to performance.
- It is interesting that traditional self-scheduling schemes (FSS and TSS) perform slightly better than NGSS and ANGSS. However, this result is inconsistent with that of previous research. The reason might be that the parameter α is set too high, 75. If the parameter α is set appropriately, it is possible for NGSS and ANGSS to perform better, as previous work has shown.

4.2 Application 2: Association Rule Mining

In this application, the workload is the dataset to be mined on. We implemented the Apriori algorithm, and applied our approach to conduct data distribution. Specifically, the parallelized version of Apriori we adopt is Count Distribution (CD) [12, 13]. In this experiment, "cd_eq" means to distribute the workload to slaves equally, and "cd_cpu" means to distribute the workload to slaves according to the ratio of CPU speed values of slaves. And, cd_pwd is the proposed scheme. Our datasets are generated by the tool as in [13]. The parameters of the synthetic datasets are described in Table 2.

Table 2. Description of our dataset

Dataset	Number of Transactions	Average Transaction Length	Number of Items
D10KT5I10	10,000	5	10
D50KT5I10	50,000	5	10
D100KT5I10	100,000	5	10
D200KT5I10	200,000	5	10

First, execution time on the grid for the three schemes is investigated. As shown in Figure 4, cd_pwd outperforms cd_eq and cd_cpu. From this experiment, we can see the significant influence of partition schemes on the total completion time. In grid environments, network bandwidth is an important criterion to evaluate the performance of a slave node. Cd_eq and cd_cpu are static data partition schemes. Therefore, they can not adapt to the practical network status. When communication cost becomes a major factor, the proposed scheme would be well adaptive to the dynamic network environment.

Moreover, the reason why cd_cpu got the worst performance can be contributed to the inappropriate estimation of node performance. In grid computing environments, CPU speed is not the only factor to determine the node performance. A node with the fastest CPU is not necessary the node with optimal performance.

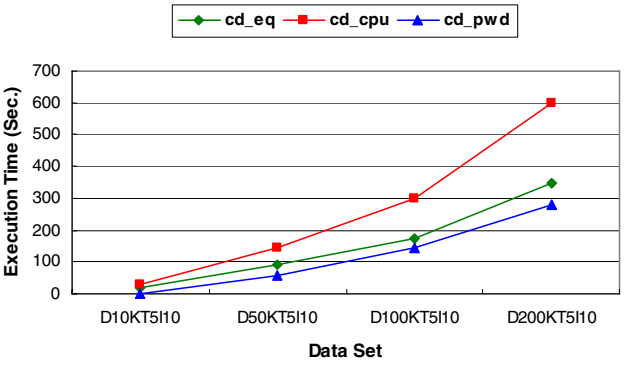

Fig. 4. Performance of data partition schemes for different datasets

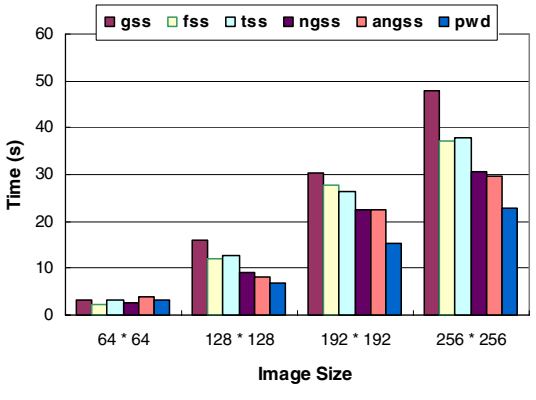

Fig. 5. Execution time for Mandelbrot Set Computation with different input sizes

4.3 Application 3: Mandelbrot Set Computation

The Mandelbrot set computation is a problem involving the same computation on different data points which have different convergence rates [34]. In the following experiment, we want to compare the execution time of previous schemes with the proposed approach. Figure 5 illustrates the results for input image size 64×64, 128×128, 192×192 and 256×256 respectively. The execution time of weighted static scheduling is omitted due to its bad performance. According to the experience in Matrix Multiplication example, the parameter α is set to 30. The results are discussed as follows.

- Among these schemes, the PWD still performs better than other schemes. The reason is also that PWD accurately estimates the PR, and takes the advantage of static scheduling, thus reducing the runtime overhead.
- Traditional self-scheduling schemes (GSS, FSS and TSS) perform worse than NGSS and ANGSS. The reason is that irregular workload is difficult to schedule. If the parameter α is set appropriately, it is certain for NGSS and ANGSS to perform better, as previous work has shown.

5 Conclusions

In this paper, we have investigated the workload distribution problem on dynamic and heterogeneous grid environments. First, a performance-based approach was proposed to schedule workloads on grid environments. In this approach, the system heterogeneity is estimated by performance functions, and the variation of workload is estimated by Static-Workload Ratio. On our grid platform, the proposed approach can obtain performance improvement on previous schemes. In our future work, we will implement more types of application programs to verify our approach.

References

[1] Divisible Load Theory, http://www.ee.sunysb.edu/~tom/MATBE/index.html
[2] Global Grid Forum, http://www.ggf.org/
[3] Introduction to Grid Computing with Globus, http://www.ibm.com/redbooks
[4] KISTI Grid Testbed, http://Gridtest.hpcnet.ne.kr/
[5] MPICH, http://www-unix.mcs.anl.gov/mpi/mpich/
[6] MPICH-G2, http://www.hpclab.niu.edu/mpi/
[7] Network Weather Service, http://nws.cs.ucsb.edu/
[8] Sun ONE Grid Engine, http://wwws.sun.com/software/Gridware/
[9] TeraGrid, http://www.teragrid.org/
[10] The Globus Project, http://www.globus.org/
[11] TIGER Grid Report, http://gamma2.hpc.csie.thu.edu.tw/ganglia/
[12] R. Agrawal and J. C. Shafer, "Parallel Mining of Association Rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 962-969, Dec. 1996.
[13] R. Agrawal and R. Srikant, "Fast algorithms for Mining Association Rules," *Proc. 20th Very Large Data Bases Conf.*, pp. 487-499, 1994.
[14] M. A. Baker and G. C. Fox. "Metacomputing: Harnessing Informal Supercomputers." *High Performance Cluster Computing*. Prentice-Hall, May 1999. ISBN 0-13-013784-7.
[15] I. Banicescu, R. L. Carino, J. P. Pabico, and M. Balasubramaniam, "Overhead Analysis of a Dynamic Load Balancing Library for Cluster Computing," *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium,* 2005.
[16] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor platforms," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 4, pp. 319-330, Apr. 2004.
[17] O. Beaumont, H. Casanova, A. Legrand, Y. Robert and Y. Yang, "Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems," IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 3, pp. 207-218, Mar. 2005.

[18] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Press, 1996.

[19] V. Bharadwaj, D. Ghose and T.G. Robertazzi, "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems," *Cluster Computing*, vol. 6, no. 1, pp. 7-18, Jan. 2003.

[20] K. W. Cheng, C. T. Yang, C. L. Lai, and S. C. Chang, "A Parallel Loop Self-Scheduling on Grid Computing Environments," *Proceedings of the 2004 IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pp. 409-414, KH, China, May 2004.

[21] A. T. Chronopoulos, R. Andonie, M. Benche and D.Grosu, "A Class of Loop Self-Scheduling for Heterogeneous Clusters," *Proceedings of the 2001 IEEE International Conference on Cluster Computing*, pp. 282-291, 2001.

[22] A. T. Chronopoulos, S. Penmatsa, J. Xu and S.Ali, "Distributed Loop-Self-Scheduling Schemes for Heterogeneous Computer Systems," *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 771-785, 2006.

[23] N. Comino and V. L. Narasimhan, "A Novel Data Distribution Technique for Host-Client Type Parallel Applications," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 2, pp. 97-110, Feb. 2002.

[24] M. Drozdowski and M. Lawenda, "On Optimum Multi-installment Divisible Load Processing in Heterogeneous Distributed Systems," *Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, Lecture Notes in Computer Science*, vol. 3648, pp. 231-240, Springer-Verlag, August 2005.

[25] I. Foster, N. Karonis, "A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems." *Proc. 1998 SC Conference*, November, 1998.

[26] I. Foster, C. Kesselman., "Globus: A Metacomputing Infrastructure Toolkit," *International J. Supercomputer Applications*, 11(2):115-128, 1997.

[27] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, 15(3), 2001.

[28] I. Foster, "The Grid: A New Infrastructure for 21st Century Science." *Physics Today*, 55(2):42-47, 2002.

[29] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001.

[30] J. Herrera, E. Huedo, R. S. Montero, and I. M. Llorente, "Loosely-coupled loop scheduling in computational grids," *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium,* 2006.

[31] S. F. Hummel, E. Schonberg, and L. E. Flynn, "Factoring: a method scheme for scheduling parallel loops," *Communications of the ACM*, Vol. 35, 1992, pp. 90-101.

[32] `C. Kruskal and A. Weiss, "Allocating independent subtaskson parallel processors," *IEEE Transactions on Software Engineering*, vol. 11, pp 1001–1016, 1984.

[33] H. Li, S. Tandri, M. Stumm and K. C. Sevcik, "Locality and Loop Scheduling on NUMA Multiprocessors," *Proceedings of the 1993 International Conference on Parallel Processing*, vol. II, pp. 140-147, 1993.

[34] B. B. Mandelbrot, *Fractal Geometry of Nature*, W. H. Freeman: New york, 1988.

[35] C. D. Polychronopoulos and D. Kuck, "Guided Self-Scheduling: a Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Trans. on Computers*, vol. 36, no. 12, pp 1425-1439, 1987.

[36] T.G. Robertazzi, "Ten Reasons to Use Divisible Load Theory," *Computer*, vol. 36, no. 5, pp. 63-68, May 2003.

[37] W. C. Shih, C. T. Yang, and S. S. Tseng, "A Performance-Based Parallel Loop Self-Scheduling on Grid Environments," *Network and Parallel Computing: IFIP International Conference, NPC 2005, Lecture Notes in Computer Science*, vol. 3779, pp. 48-55, Springer-Verlag, December 2005.

[38] W. C. Shih, C. T. Yang, and S. S. Tseng, "A Hybrid Parallel Loop Scheduling Scheme on Grid Environments," *Grid and Cooperative Computing: 4th International Conference, GCC 2005, Lecture Notes in Computer Science*, vol. 3795, pp. 370-381, Springer-Verlag, December 2005.

[39] W. C. Shih, C. T. Yang, and S. S. Tseng, "A Performance-based Approach to Dynamic Workload Distribution for Master-Slave Applications on Grid Environments," *GPC 2006, Lecture Notes in Computer Science*, vol. 3947, pp. 73-82, Springer-Verlag, 2006.

[40] L. Smarr, C. Catlett, "Metacomputing," *Communications of the ACM*, vol. 35, no. 6, pp. 44-52, 1992.

[41] S. Tabirca, T. Tabirca and L. T. Yang, "A convergence study of the discrete FGDLS algorithm," *IEICE Transactions on Information and Systems*, vol. E89-D, no. 2, pp. 673-678, 2006.

[42] P. Tang and P. C. Yew, "Processor self-scheduling for multiple-nested parallel loops," Proceedings of the 1986 International Conference on Parallel Processing, pp. 528-535, 1986.

[43] T. H. Tzen and L. M. Ni, "Trapezoid self-scheduling: a practical scheduling scheme for parallel compilers," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, 1993, pp. 87-98.

[44] C. T. Yang and S. C. Chang, "A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters," *Journal of Information Science and Engineering*, vol. 20, no. 2, pp. 263-273, March 2004.

[45] C. T. Yang, K. W. Cheng, and K. C. Li, "An Efficient Parallel Loop Self-Scheduling on Grid Environments," *NPC'2004 IFIP International Conference on Network and Parallel Computing, Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, Hai Jin, Guangrong Gao, Zhiwei Xu (Eds.), Oct. 2004.

[46] C. T. Yang, K. W. Cheng, and K. C. Li, "An Efficient Parallel Loop Self-Scheduling Scheme for Cluster Environments," *The Journal of Supercomputing*, vol. 34, pp. 315-335, 2005.

[47] M. J. Zaki, "Parallel and Distributed Association Mining: A Survey," *IEEE Concurrency*, vol. 7, no. 4, pp. 14-25, 1999.