Using a Performance-based Skeleton to Implement Divisible Load Applications on Grid Computing Environments[*]

WEN-CHUNG SHIH[1], CHAO-TUNG YANG[+] AND SHIAN-SHYONG TSENG[1,2]
[1]*Department of Information Science and Applications*
Asia University
Taichung, 413 Taiwan
E-mail: {wjshih; sstseng}@asia.edu.tw
[+]*High-Performance Computing Laboratory*
Department of Computer Science and Information Engineering
Tunghai University
Taichung, 407 Taiwan
E-mail: ctyang@thu.edu.tw
[2]*Department of Computer Science*
National Chiao Tung University
Hsinchu, 300 Taiwan
E-mail: sstseng@cis.nctu.edu.tw

Applications with divisible loads have such a rich source of parallelism that their parallelization can significantly reduce their total completion time on grid computing environments. However, it is a challenge for grid users, probably scientists and engineers, to develop their applications which can exploit the computing power of the grid. We propose a performance-based skeleton algorithm for implementing divisible load applications on grids. Following this skeleton, novice grid programmers can easily develop a high performance grid application. To examine the performance of programs developed by this approach, we apply this skeleton to implement three kinds of applications and conduct experiments on our grid test-bed. Experimental results show that programs implemented by this approach run more rapidly than those using conventional scheduling schemes.

Keywords: divisible load application, workload distribution, grid computing, message passing interface, parallel programming

1. INTRODUCTION

As computers become more and more inexpensive and powerful, computational grids which consist of various computational and storage resources have become promising alternatives to traditional multiprocessors and computing clusters [1, 2]. Basically, grids are distributed systems which share resources through the internet. On the one hand, users can access more computing resources through grid technologies. On the other hand, grid environments require effective management to operate in an efficient way. Moreover, the heterogeneity and dynamic changing of the grid environment make it different from conventional parallel and distributed computing systems, such as multiprocessors

Received April 19, 2007; revised December 24, 2007; accepted January 31, 2008.
Communicated by Chung-Ta King.
[*] The work was supported in part by the National Science Council of Taiwan, R.O.C., under grants No. NSC 96-2221-E-029-019-MY3 and NSC 97-2511-S-468-003.
[+] Corresponding author.

and computing clusters. Therefore, it is a challenge to utilize the grid efficiently.

Applications with divisible loads are a rich source of parallelism. Programmers can identify independent work units within a program and dispatch them to different processors to reduce its completion time. Nowadays, parallelizing a program for grid platforms mainly depends on human efforts. Automatic transformation of parallel applications into Grid-aware ones was investigated in [3-5], but their approach is not suitable for a novice programmer to develop parallel applications from scratch. Furthermore, it is difficult for programmers to acquire real-time grid status information and to appropriately distribute workload within a program to heterogeneous working nodes.

Our idea is to provide programmers with a template program, which takes care of details related to grid infrastructure. All the programmers need to do is to fill in the skeleton algorithm with application-specific code fragments. The resulting program can appropriately distribute the workload of the program to working nodes according to dynamic node performance. That is, we propose a performance-based skeleton algorithm, which serves as a template for programmers to develop a parallel program. To verify this approach, we apply this skeleton to three types of applications, Matrix Multiplication, Association Rule Mining and Mandelbrot Set Computation, and execute them in a grid test-bed. Experimental results show that programs developed by this approach can exploit the computing power of the grid.

The primary advantage of this approach is that a programmer can easily develop high performance programs to execute on grid environments. The high performance results from two features of this skeleton. First, it is a hybrid method. In its first phase, workload is distributed statically according to node performance to reduce scheduling overhead. In the second phase, the remaining load is dispatched dynamically to achieve load balance. Second, it utilizes real-time information to estimate the performance of the grid. The skeleton acquires grid status information from a monitoring tool and estimates the performance of computing and communication resources with the information.

Our contributions can be summarized as follows. First, this paper proposes a performance-based skeleton for programmers to develop high-quality parallel applications with ease. Programs developed by this approach can utilize grid information to adaptively distribute workloads within a program. Second, we apply this skeleton to three kinds of divisible load applications on our grid test-bed. Consequently, experimental results show the obvious effectiveness of our approach. Note that this work aims at a general skeleton of workload distribution, instead of proposing a new loop scheduling scheme or a novel data mining algorithm.

The remainder of this paper is organized as follows. In section 2, divisible load theory and dynamic loop scheduling schemes are reviewed. In section 3, we describe the proposed approach to developing a performance-based parallel application. Next, the configuration of our grid test-bed 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. RELATED WORK

In this section, the theory of divisible load is briefly reviewed. Then, we present some well-known loop scheduling schemes.

2.1 Divisible Load Theory

Divisible Load Theory (DLT) addresses the case where the total workload can be partitioned into any number of independent sub-jobs. In the past, the theory of divisible load has been widely investigated in static heterogeneous systems. However, it has not been widely applied to computing grids, which are characterized by heterogeneous resources and dynamic environments. This problem has been discussed in the past decade, and a good review can be found in [6]. In [7, 8], an exact method for divisible load was proposed, which was not from a dynamic and pragmatic viewpoint as ours. DLT focuses on coarse-grain loads, which are a pool of jobs or programs. However, the target of this work is fine-grain loads, which might be loop iterations within a program, for example. We focus on the problem of parallelizing an application with divisible loads for rapid execution on grid environments. Since grid environments are dynamically changing and heterogeneous, the problem is obviously different from the traditional DLT problem.

2.2 Loop Scheduling Schemes

Conventionally, loop scheduling schemes are classified according to the time when the scheduling decision is made. Static loop scheduling schemes make a scheduling decision at compile time, and equally assign the total iterations of a loop to processors. It is applied when each iteration of a loop takes roughly the same amount of time, and the compiler knows enough related information before compilation. Its advantage is less overhead at runtime, while the disadvantage is possible load imbalance. Well-known static scheduling schemes include Block Scheduling, Cyclic Scheduling, Block-D Scheduling, Cyclic-D Scheduling, *etc*. However, these schemes are not suitable for dynamic grid environments.

Dynamic loop scheduling schemes make a scheduling decision at runtime. Its disadvantage is more overhead at runtime, while the advantage is load balance. Several self-scheduling schemes are restated here as follows.

Pure Self-scheduling (PSS) This is a straightforward dynamic loop scheduling algorithm [9]. 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. When the chunk size is one, this scheme is PSS, as discussed above. If the chunk size is set to the bound of the parallel loop equally divided by the number of processors, this scheme becomes static scheduling. A large chunk size will cause load imbalance while a small chunk size is likely to result in too much runtime overhead.

Guided Self-scheduling (GSS) This scheme can dynamically change the number of iterations assigned to each processor [10]. 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. The property of decreasing chunk size implies an effort is made to achieve load balance and to reduce the runtime overhead. By assigning large chunks at the beginning of a parallel loop, one can reduce the frequency of communication between the master and slaves.

Factoring Self-scheduling (FSS) In some cases, GSS might assign too much work to the first few processors, so that the remaining iterations are not time-consuming enough to balance the workload. The Factoring algorithm addresses this problem [11]. 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. Therefore, it balances loads better than GSS does when the computation times of loop iterations vary substantially. In addition, the synchronization overhead of Factoring is not significantly larger than that of GSS.

Trapezoid Self-scheduling (TSS) This approach tries to reduce the need for synchronization while still maintaining a reasonable load balance [12]. TSS(N_s, N_f) assigns the first N_s iterations of a loop to the processor starting the loop and the last N_f iterations to the processor performing the last fetch, where N_s and N_f are both specified by the programmer or the system. This algorithm allocates large chunks of iterations to the first few processors and successively smaller chunks to the last few processors. Tzen and Ni proposed TSS($N/2p$, 1) as a general selection.

Table 1 shows the chunk sizes for the self-scheduling schemes above with respect to a loop with 1000 iterations. Besides, the number of available processors is 4.

Table 1. Sample partition size.

Scheme	Sample partition size
PSS	1, 1, 1, 1, 1, 1, 1, 1, 1, …
CSS(125)	125, 125, 125, 125, 125, 125, 125, 125
FSS	125, 125, 125, 125, 63, 63, 63, 63, 31, …
GSS	250, 188, 141, 106, 79, 59, 45, 33, 25, …
TSS	125, 117, 109, 101, 93, 85, 77, 69, 61, …

In [13], 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. For example, GSS can be enhanced by partitioning α percent of the total iterations according to their performance weighted by CPU clock in the first phase. Then, the remainder of the workload is still scheduled by GSS. This enhanced scheme is called NGSS.

In [14], 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.

Our previous work [15, 16] presents different heuristics to the parallel loop self-scheduling problem. This paper extends the idea of performance-based scheduling to design a performance-based skeleton for developing high performance applications on grids. This approach is applied to both the parallel loop self-scheduling application and the association rule mining application.

3. APPROACH

In this section, the system model is introduced 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 modern Grid paradigm consists of clusters that are controlled by local schedulers. Also, there are meta-schedulers which have a global view of the whole infrastructure. However, the viewpoint of a user application is simpler, which considers how many resources it can use. Therefore, the abstract view of the system 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. For example, as shown in Fig. 1, N is $\{P_0, P_1, \ldots, P_n\}$ and E is $\{L_1, L_2, \ldots, L_n\}$. In this example, P_0 is the master node and the other n nodes, $P_1, \ldots, P_n$, are slave nodes. Conceptually, there is a virtual link L_i connecting the master node and a slave node P_i. In reality, L_i may be composed of several networking segments connected by switches or/and routers.

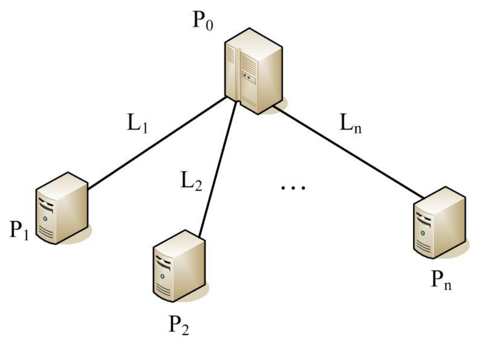


Fig. 1. Abstract overview of the system model.

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 variable attributes. In the following sections, the two kinds of attributes are utilized to model the heterogeneity of a dynamic grid.

Programming models are generally classified by the way memory is used. In the shared memory model each process accesses a shared address space, while in the message passing model processes communicate with other processes by sending and receiving messages. The message-passing paradigm is adopted in this paper. Basically, the programmer assumes the system consists of several processors, each with its own memory space, and writes a program to run on each processor. However, parallel programming generally requires communication between the processors to complete a task. The characteristic of the message-passing paradigm is that the processors communicate by sending messages instead of shared memory. Therefore, in the message-passing model, processors can not access each other's memory directly.

3.2 Performance Ratio

The concept of performance ratio was previously defined in [15, 16] in different forms and parameters, according to the requirements of applications. In this work, the skeleton algorithm uses a performance function to model the heterogeneous performance of the dynamic grid nodes. The purpose of calculating performance ratio is to estimate the current processing capability for each node. With this metric, the program can distribute appropriate workloads to each node, and load balance can be achieved. The more accurate the estimation is, the better the load balance is.

Assume that m is the number of attributes. For example, this study adopts three attributes: CPU speed, CPU loading, and Bandwidth. Therefore, m is equal to 3. To estimate the performance of each slave node, a performance function (PF) is defined for a slave node j:

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

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, TIGER [17], to acquire the values of attributes for all slaves. 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 available 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 TIGER tool.
- CL_i is the CPU loading of node i, and it is a variable attribute. The value of this parameter is acquired by the TIGER tool.
- B_i is the bandwidth (Mbps) between node i and the master node. The value of this parameter is also acquired by the TIGER tool.
- w_1 and w_2 are the weights of the first and second term, respectively. The sum of the two parameters is equal to one. The values of the two parameters are decided by experiments on different combinations of the two parameter values. The combination with the best performance is adopted for actual use.

The performance ratio (PR) is defined to be the ratio of all performance functions. For instance, assume the PF values of three nodes are 1/2, 1/3 and 1/4. Then, the PR is 1/2 : 1/3 : 1/4; *i.e.*, the PR of the three nodes is 6 : 4 : 3. In other words, if there are 13 loop iterations, 6 iterations will be assigned to the first node, 4 iterations will be assigned to the second node, and 3 iterations will be assigned to the last one.

3.3 Determination of Static-Workload Ratio (SWR)

Another important factor to be estimated is the variation degree among all units of workloads. For example, Mandelbrot Set Computation is a problem involving irregular workloads. In each iteration of a loop, the workload is different and varies significantly, as shown in Fig. 2. Obviously, a distribution scheme which does not consider the effect of irregular workload could not estimate PR accurately.

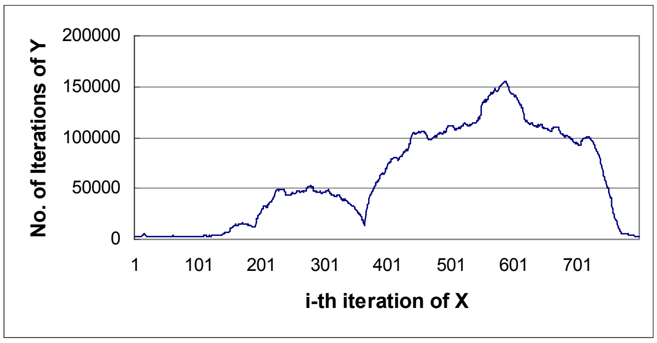


Fig. 2. The Mandelbrot set on [– 1.8, 0.5] to [– 1.2, 1.2] an 800 × 800 pixel window.

We propose to use a parameter, SWR (Static-Workload Ratio), ranging from 0 to 1, to estimate the proportion of the workload which can be statically scheduled, alleviating the effect of irregular workload. In order to take advantage of static scheduling, the *SWR* proportion of the total workload is dispatched according to Performance Ratio. The design rationale is based on a conservative heuristic to estimate the irregular degree of workloads among all iterations. If the workload of the target application is regular, SWR can be set to be 1. 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 value for 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 1, 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 10 seconds, respectively. Then the SWR is 7/10. Therefore, 70% of the workload would be scheduled statically according to PR, while 30% of the workload would be scheduled by a dynamic scheme.

3.4 The Skeleton Algorithm

Based on the estimated information of workload distribution and node performance, we propose a skeleton algorithm for performance-based workload distribution on grid environments. This algorithm is based on a message-passing paradigm, and consists of two modules: a master module and a slave module. The master module makes the scheduling decision and dispatches workloads to slaves. On the other hand, the slave module processes the assigned work. This algorithm is just a skeleton, and the detailed implementation, such as data preparation, parameter passing, *etc*., might be different according to requirements of various applications.

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)-percent 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 scheduled by a dynamic scheme for load balancing. The algorithm of our approach is described as follows.

```
Module MASTER
Initialization
/* Stage 1: Gathering the information */
   collect the following information from the TIGER tool:
      – CPU_Loading
      – CPU_Clock_Speed
      – Network_Bandwidth
   collect the execution time of 5 sampled iterations

/* Stage 2: Calculate two scheduling parameters */
   calculate SWR of the workload
   calculate Performance Ratio of all slave nodes

/* Stage 3: Static Scheduling */
   dispatch the (SWR)-percent of workload according to Performance Ratio
   probe and receive for returned results

/* Stage 4: dynamic Scheduling */
   dispatch the (100-SWR)-percent of workload by a dynamic scheme

Finalization
END MASTER
```

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

4. EXPERIMENTAL RESULTS

To verify our approach, a grid test-bed was built, and three types of application programs were implemented using the skeleton: Matrix Multiplication, Association Rule Mining and Mandelbrot Set Computation. The former two applications have regular workloads, while the last has irregular workload.

4.1 Grid Test-bed: TIGER Project

A metropolitan-scale Grid computing platform named TIGER Grid [17] (standing for Taichung Integrating Grid Environment and Resource) has been built in a project leaded by Tunghai University. The TIGER grid interconnects computing resources of universities and high schools and shares available resources among them, for investigations in system technologies and high performance applications. This novel project shows the viability of implementation of such a project in a metropolitan city. The participating schools of the TIGER Grid computing platform are all located in Taichung, Taiwan. The project of constructing such a grid infrastructure was to share computational resources of each institution.

We have built a grid test-bed based on part of the TIGER Grid, using the following middleware:

- Globus Toolkit 4.0.2 [2, 18].
- MPICH-G2 library 1.2.6 [19].

The master node is at Tunghai University (THU), and the slave nodes are located at Tunghai University (THU), Providence University (PU), Li-Zen High School (LZ), and Hsiuping Institute of Technology School (HIT). Fig. 3 shows our grid test-bed, and the specifications of the grid test-bed are shown in Table 2. Fig. 4 shows the real-time status of the grid test-bed acquired by the monitoring tool.

In this study, we have implemented several scheduling schemes for the purpose of evaluation. For readability of experimental results, the brief description of all implemented schemes is listed in Table 3.

The conventional static scheduling scheme is to equally distribute the total workload to each worker at compile time. However, this scheme is obviously not suitable for dynamic and heterogeneous environments. Therefore, a weighted static scheduling scheme

Table 2. Specifications of computing resources on the test-bed.

Site	Host	CPU Type	Clock (Mhz)	RAM	NIC	Linux Kernel	Globus Version
THU	delta1	Intel Pentium 4	3001	1GB	1G	2.6.12	4.0.1
	delta2	Intel Pentium 4	3001	1GB	1G	2.6.12	4.0.1
	delta3	Intel Pentium 4	3001	1GB	1G	2.6.12	4.0.1
	delta4	Intel Pentium 4	3001	1GB	1G	2.6.12	4.0.1
LZ	lz01	Intel Celeron	898	256MB	10/100	2.4.20	4.0.1
	lz02	Intel Celeron	898	256MB	10/100	2.4.20	4.0.1
	lz03	Intel Celeron	898	384MB	10/100	2.4.20	4.0.1
	lz04	Intel Celeron	898	256MB	10/100	2.4.20	4.0.1
HIT	gridhit0	Intel Pentium 4	2800	512MB	10/100	2.6.12	4.0.1
	gridhit1	Intel Pentium 4	2800	512MB	10/100	2.6.12	4.0.1
	gridhit2	Intel Pentium 4	2800	512MB	10/100	2.6.12	4.0.1
	gridhit3	Intel Pentium 4	2800	512MB	10/100	2.6.12	4.0.1
PU	hpc09	AMD Athlon XP	1991	1GB	1G	2.4.22	4.0.1
	hpc10	AMD Athlon XP	1991	1GB	1G	2.4.22	4.0.1
	hpc11	AMD Athlon XP	1991	1GB	1G	2.4.22	4.0.1
	hpc12	AMD Athlon XP	1991	1GB	1G	2.4.22	4.0.1

Table 3. Description of all implemented programs.

Scheduling Scheme	Description	Reference
static	Weighted static scheduling	
gss	Dynamic scheduling (GSS)	[10]
fss	Dynamic scheduling (FSS)	[11]
tss	Dynamic scheduling (TSS)	[12]
ngss	Fixed α scheduling + GSS	[13]
angss	Adaptive α scheduling + GSS	[14]
pwd	Proposed scheduling	

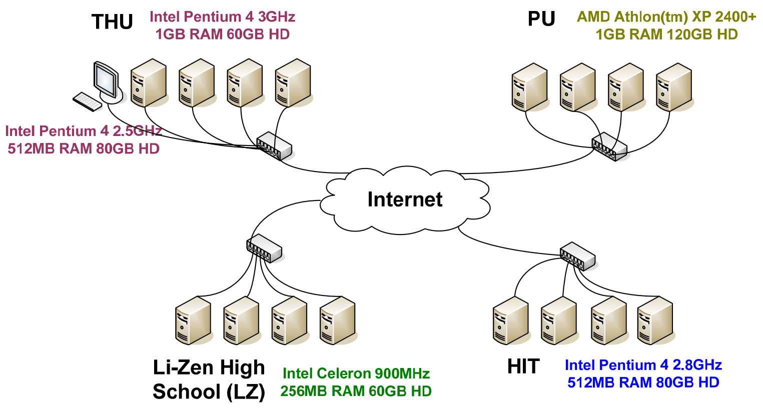
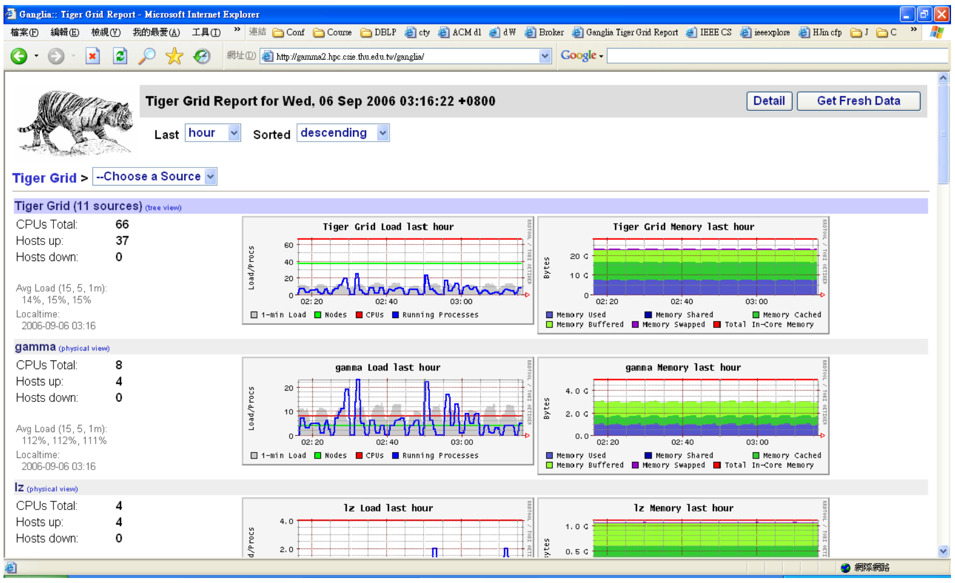


Fig. 3. The logical diagram of our grid test-bed.


Ganglia:: Tiger Grid Report - Microsoft Internet Explorer

http://gamma2.hpc.csie.thu.edu.tw/ganglia/

Tiger Grid Report for Wed, 06 Sep 2006 03:16:22 +0800

Last hour Sorted descending

Tiger Grid > --Choose a Source

Tiger Grid (11 sources) (tree view)

CPUs Total: 66
Hosts up: 37
Hosts down: 0

Avg Load (15, 5, 1m):
14%, 15%, 15%
Localtime:
2006-09-06 03:16

gamma (physical view)

CPUs Total: 8
Hosts up: 4
Hosts down: 0

Avg Load (15, 5, 1m):
112%, 112%, 111%
Localtime:
2006-09-06 03:16

lz (physical view)

CPUs Total: 4
Hosts up: 4
Hosts down: 0

Fig. 4. The snapshot of the monitoring tool on the TIGER grid.

is adopted in this experiment. The principle of partitioning is according to the CPU clock speed of each processor. A faster node will get more workloads than a slower one proportionally.

To reduce errors of experimental results, execution time in each experiment is obtained by averaging the results of five repetitive executions.

4.2 Application 1: Matrix Multiplication

Matrix Multiplication is a fundamental operation in many numerical linear algebra applications. Its efficient implementation on parallel computers is an issue of prime importance when providing such systems with scientific software libraries. Consequently, considerable effort has been devoted in the past to the development of efficient parallel matrix multiplication algorithms, and this will remain a task in the future as well. Many parallel algorithms have been designed, implemented, and tested on different parallel computers or cluster of workstations for matrix multiplication.

In this application, the workload is loop iterations. The Master module is responsible for the distribution of workloads. When a slave node becomes idle, the master node sends two integers to the slave. The two numbers represent the beginning and ending pointers to the assigned chunk respectively. In other words, every node has a copy of the input matrices locally, so data communication is not significant in this kind of implementation. Therefore, communication cost between the master and the slave is low, and the dominant cost is the computation of matrix multiplication. The C/MPI code fragment of the Slave module for Matrix Multiplication is listed as follows. As the source code shows, a column is the atomic unit of allocation.

```
MPI_Recv(buf, count, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &status);
f = 0;
while (status.MPI_TAG > 0)
{
for (i = 0; i < (count/SIZE); i++)
    for (j = 0; j < SIZE; j++)
        c[i * SIZE + j] = 0.0;

    /* computing */
    for (i = 0; i < (count/SIZE); i++)
        for (j = 0; j < SIZE; j++)
        for (k = 0; k < SIZE; k++)
            c[i * SIZE + j] += buf[i * SIZE + k] * b[k * SIZE + j];

    /* sent result*/
    MPI_Send(c, count, MPI_FLOAT, 0, tag, MPI_COMM_WORLD);
    free(buf);
    free(c);

    /* get another size */
    MPI_Probe(0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    source = status.MPI_SOURCE;
    tag = status.MPI_TAG;
    MPI_Get_count(&status, MPI_FLOAT, &count);
    buf = (float*)malloc(count * sizeof(float));
    c = (float*)malloc(count * sizeof(float));
    MPI_Recv(buf, count, MPI_FLOAT, 0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    }
}
```

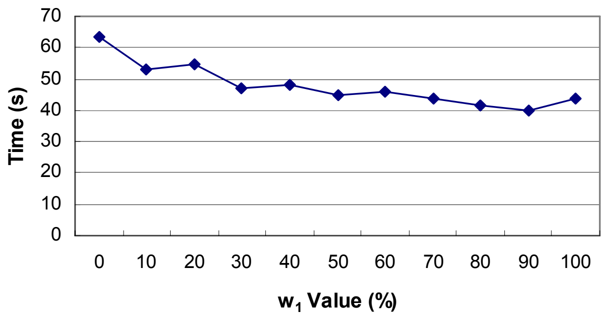


Fig. 5. Execution time for matrix multiplication with different values of parameters.

The appropriate values for w_1 and w_2 in Eq. (2) are determined by the following experiment. Fig. 5 depicts the execution time of our PWD scheme for input matrix size 1024 × 1024, with w_1 set from 0 to 100 percent. When w_1 is 90 percent, the execution is minimal in this experiment. The reason might be that the communication cost is low in this program. Therefore, we adopt 90 and 10 as the w_1 and w_2 value, respectively.

First, we want to compare the proposed PWD scheme with previous schemes with respect to the execution time. Fig. 6 illustrates the execution time of weighted static scheduling, GSS, FSS, TSS, NGSS, ANGSS and our PWD scheme, with input matrix size 512 × 512, 1024 × 1024, 1536 × 1536 and 2048 × 2048 respectively. The results are shown as follows.

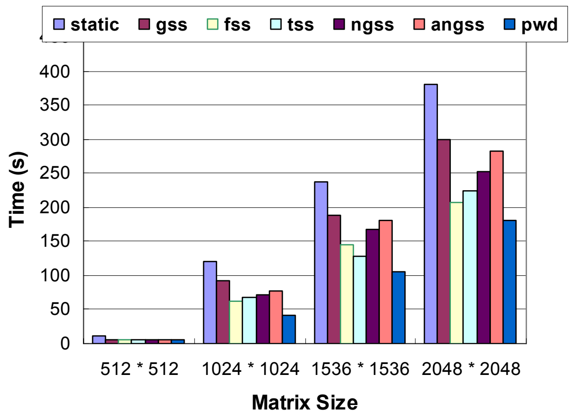


Fig. 6. Execution time for matrix multiplication with different input sizes.

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 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 [13, 14]. 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. This case also indicates that NGSS and ANGSS suffer from the difficulty of determining an appropriate parameter value.

4.3 Application 2: Association Rule Mining

Data mining, or known as knowledge discovery, is to acquire interesting knowledge from large-scale databases [20]. Data mining techniques include association rule mining, classification, cluster analysis, *etc*. The objective of association rule mining is to discover correlation relationships among a set of items. The well-known application of association rule mining is market basket analysis. This technique can extract customer buying behaviors by discovering 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 [21, 22]. Let I 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 $A \cap B = \varnothing$. 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 [23]. Traditional parallel data mining work assumes data is partitioned and transmitted to the computing nodes in advance. However, it is usually the case in which a large database is generated and stored in some station. Therefore, it is important to efficiently partition and to distribute the data to other nodes for parallel computation.

In this application, the workload is a database of transactions. We applied the skeleton to implement the Apriori algorithm and its data distribution. Specifically, the parallelized version of Apriori we adopt is Count Distribution (CD) [21, 22]. Our datasets are generated by the tool indicated in [22]. The parameters of the synthetic datasets are described in Table 4.

Table 4. 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
D150KT5I10	150,000	5	10

The C/MPI code fragment of the Slave module for Count Distribution is listed as follows. For simplicity, only the first two frequent set computations are shown.

```
MPI_Status status;
// MPI_Request request;

    /* receive data from master at first time */
    MPI_Probe(0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    source = status.MPI_SOURCE;
    tag = status.MPI_TAG;
    MPI_Get_count(&status, MPI_INT, &count);
    MPI_Recv(&db[1][0], count, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
    // Large 1 computation
    // initialize local_L1 to 0
    for (i = 0; i ≤ N_item; i++) local_L1[i] = 0;
    // count local # items
    for (i = 1; i ≤ count/LENGTH; i++)
    {
        for (j = 1; j ≤ db[i][0]; j++)
        {
            local_L1[db[i][j]]++;
        }
    }
        MPI_Reduce(local_L1, large_1, N_item+1, MPI_INT, MPI_SUM, 0, MPI_COMM_
        WORLD);
    MPI_Barrier(MPI_COMM_WORLD);

    // Large 2 computation
    // initialize local_L2 to 0
    for (i = 0; i ≤ N_item * N_item; i++) local_L2[i] = 0;
    // count local # 2-items
    for (i = 1; i ≤ count/LENGTH; i++)
    {
        for (j = 1; j ≤ db[i][0] – 1; j++)
        {
            for (k = j + 1; k ≤ db[i][0]; k++)
            {
                local_L2[(db[i][j]) * N_item + db[i][k]]++;
            }
        }
    }

    MPI_Reduce(local_L2, large_2, N_item*N_item+1, MPI_INT, MPI_SUM, 0, MPI_COMM_
    WORLD);
    MPI_Barrier(MPI_COMM_WORLD);
    }
```

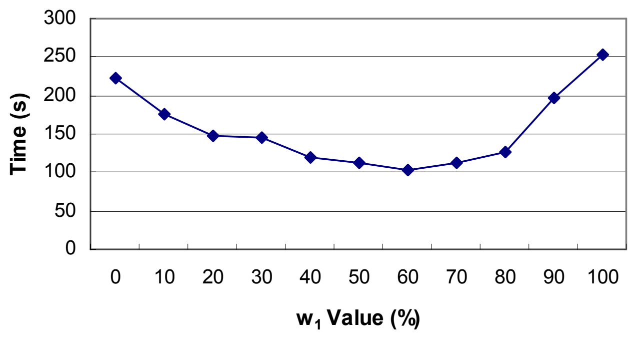


Fig. 7. Execution time for association rule mining with different values of parameters.

The appropriate values for w_1 and w_2 in Eq. (2) are decided by the following experiment. Fig. 7 shows the execution time of the proposed scheme for dataset size 50K transactions, with w_1 set from 0 to 100 percent. When w_1 is 60 percent, the execution is minimal in this experiment. The reason might be that the communication cost is higher than that of Matrix Multiplication. Therefore, we adopt 60 and 40 as the w_1 and w_2 value, respectively.

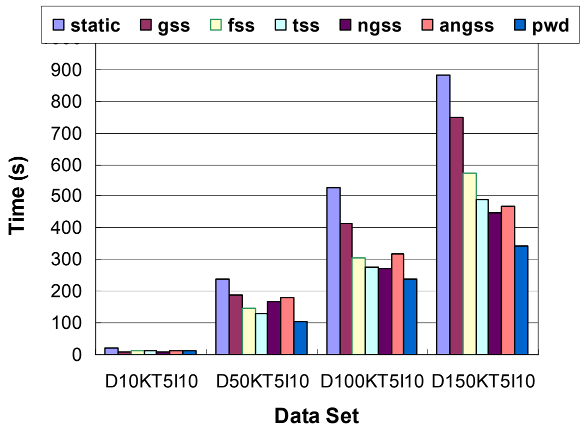


Fig. 8. Performance of data partition schemes for different datasets.

Fig. 8 illustrates the execution time of different scheme, with input size 10K, 50K, 100K and 150K transactions respectively. Experimental results show that the scheme implemented by the skeleton got better performance than other schemes.

From this experiment, we can see the significant influence of workload distribution schemes on the total response time. In grid environments, network bandwidth is an important criterion to evaluate the performance of a slave node. The Static scheme can not adapt to the practical network status. In contrast to Static, when communication cost becomes a major factor, dynamic schemes would be well adaptive to the network environment.

Moreover, the reason why PWD got the best performance can be attributed to the appropriate estimation of node performance, especially for the attribute of network bandwidth. 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.

4.4 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 [24]. This operation derives a resultant image by processing an input matrix, A, where A is an image of *a* pixels by *b* pixels. The resultant image is one of *a* pixels by *b* pixels. The Mandelbrot Set Computation has been implemented using the skeleton. The Master module is responsible for the distribution of workload. When a slave node becomes idle, the master node sends two integers to the slaves. As implemented in Matrix Multiplication, communication cost between the master and the slave is low, and the dominant cost is the computation of the Mandelbrot Set. The C/MPI code fragment of the Slave module for Mandelbrot Set Computation is listed as follows.

```
MPI_Probe(0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
source = status.MPI_SOURCE;
tag = status.MPI_TAG;
MPI_Get_count(&status, MPI_INT, &count);
MPI_Recv(&b[0], count, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
while (status.MPI_TAG > 0) {
/* Compute pixels in parallel */

    // t1 = MPI_Wtime();
    for (i = 0; i < Nx * Ny; i++)pix_tmp[i] = 0.0;

    for (y = b[0]; y < b[1]; y++){
        for (x = 0; x < Nx; x++){
            c.real = Rx_min + ((double)x * (Rx_max – Rx_min)/(double)(Nx – 1));
            c.imag = Ry_min + ((double)y * (Ry_max – Ry_min)/(double)(Ny – 1));
            pix_tmp[y * Nx + x] = cal_pixel(c);
        } // for x
    } // for y
    /* sent result */
    MPI_Send(&b[0], count, MPI_INT, 0, tag, MPI_COMM_WORLD);
    /* get another size */
    MPI_Probe(0, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    source = status.MPI_SOURCE;
    tag = status.MPI_TAG;
    MPI_Get_count(&status, MPI_INT, &count);
    MPI_Recv(&b[0], count, MPI_INT, source, tag, MPI_COMM_WORLD,&status);
    }
```

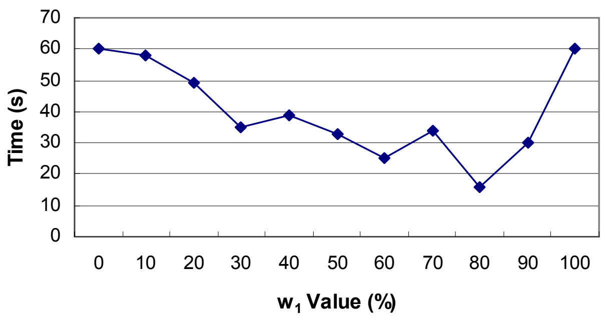


Fig. 9. Execution time for Mandelbrot set computation with different values of parameters.

The appropriate values for w_1 and w_2 in Eq. (2) are determined by the following experiment. Fig. 9 illustrates the execution time of the PWD scheme for input image size 192 × 192, with w_1 set from 0 to 100 percent. When w_1 is 80 percent, the execution is minimal in this experiment. The reason might be that the communication cost is low in this program. Therefore, we adopt 80 and 20 as the w_1 and w_2 value, respectively.

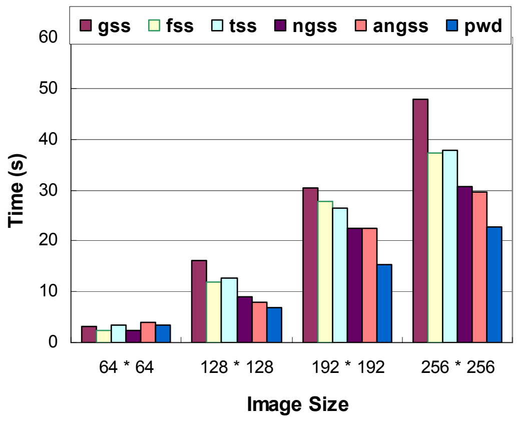


Fig. 10. Execution time for Mandelbrot set computation with different input sizes.

In the following experiment, we want to compare the execution time of previous schemes with the implemented program. Fig. 10 illustrates the execution time of GSS, FSS, TSS, NGSS, ANGSS and our PWD scheme, with input image size 64 × 64, 128 × 128, 192 × 192 and 256 × 256 respectively. The execution time of weighted static scheduling is omitted because its results are significantly inferior to other schemes. According to the experience in the Matrix Multiplication application, the parameter α in NGSS is set to 30. The results are shown as follows.

Among these schemes, 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 it is difficult to efficiently schedule irregular workload for conventional dynamic schemes. If the parameter α is set appropriately, it is certain for NGSS and ANGSS to perform better than GSS, FSS and TSS, as previous work has shown.

4.5 Discussion

In this section, several issues are discussed to clarify the proposed approach. In general, task scheduling in grid systems mainly focuses on fine grain parallelism, under the consideration of the system heterogeneity and the message-passing communication. However, one goal of grid computing is to exploit potential parallelism in internet-scale grid environments. In addition to coarse grain parallelism, we think that it is beneficial to exploit fine grain parallelism in grid systems. The first reason is to improve utilization. The proposed approach provides a mechanism for programmers to efficiently utilize the idle resources located in grid systems. The preliminary results presented in this study show that exploiting fine grain parallelism is promising. Second, the difficulties resulting from system heterogeneity and the message-passing communication can be overcome by advanced techniques, which also motivate novel research topics. Therefore, a number of researches focus on exploiting fine grain parallelism for loop scheduling and data mining in grid systems, such as [25-29].

In section 3.1, we mention that there are two kinds of attributes associated with nodes, constants and variables. It is an interesting issue to investigate the relationship between these two kinds of attributes. We think that each device in a grid system can be associated with these two kinds of attributes. Taking CPU for example, CPU clock speed is a constant attribute while CPU loading is a variable attribute. With respect to the relationship between the two kinds, it is intuitive that the node with high CPU speed will get more tasks to execute, resulting in high CPU loading. It is probable that other devices also reveal similar properties. However, this work does not focus on this topic. We plan to take this relationship into further consideration in our future work.

In this work, we primarily propose a useful grid programming skeleton, which adopts a performance-based heuristic to distribute workloads, for master-slave applications. However, we believe that it is possible to extend this approach to non-master-slave applications, such as P2P applications. We explain the reason as follows. The programming skeleton abstracts our experiences in programming master-slave applications for grid environments, which is a difficult task for novice programmers. Nevertheless, with the skeleton, all a programmer need to do is just to fill the application-specific program codes into the skeleton. If a programmer can code a sequential program, then it is straightforward to transform it to a grid application. To extend the skeleton idea to non-master-slave applications, such as P2P networks, we need to acquire experiences and expertise in P2P programming. In addition, the lack of global statistical in non-master- slave applications is a problem to be solved. In P2P networks, the performance-related information can be gathered through social activities, such as gossip protocols. This will be an interesting research topic in our future work.

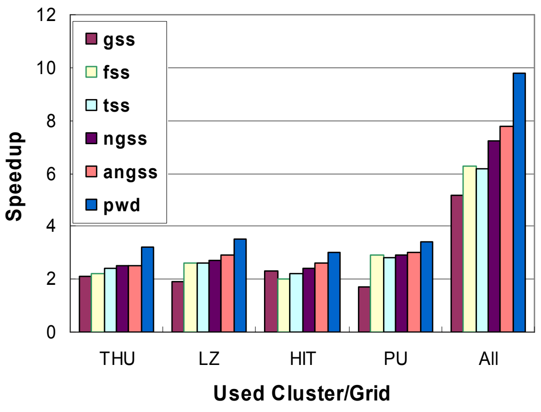


Fig. 11. Speedup for loop scheduling using different cluster or grids.

To address the performance improvement with respect to single processor, we have conducted the following experiment. The experimental setting is similar to those in section 4.2, Matrix Multiplication. Fig. 11 shows the speedup results for matrix size of 1536 × 1536. For each of THU, LZ, HIT and PU clusters, four processors participating in computation, while "all" means that 16 processors of the four cluster participating in computation. Therefore, the optimal speedup for the four clusters is 4, while the optimal speedup for the "all" configuration is 16. The result shows that the proposed approach performs better than other methods with respect to speedup. However, the speedup for the "all" configuration is only near to 10. This might result from the heterogeneity of CPU speed.

Finally, we do not mean to try all the possible values of weight (w1) in order to get high performance. Instead, we think the weight determination in this work should be application-specific. In addition, the weights for different applications should be calculated in a preprocessing phase, and be improved incrementally by a knowledge-based approach, which will be another interesting issue. In addition, we consider the weight as a pre- computed value, representing an expertise acquired from previous executions. So, it is not necessary to reflect this overhead in the timing comparison. Also, since the weight is not generated before each execution, it is not an optimal setting with respect to the next execution. However, in a dynamically changing grid environment, it is difficult to define and find an optimal solution. Therefore, the objective of the proposed heuristic algorithm is to generate a better solution than existing algorithms. The experimental results show that the proposed algorithm performs well in the dynamic grid. Therefore, we think the weight is related to the type of application, instead of problem size. That is, it is likely that the weights obtained from the same type of applications, such as computation-intensive applications, can be applied to the same type of application.

5. CONCLUSIONS

We have proposed a skeleton algorithm for programmers to easily develop high performance applications on dynamic and heterogeneous grid environments. This skeleton algorithm uses a performance-based approach to distribute workloads within a program to working nodes. 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, programs implemented by the proposed approach can obtain performance improvement on previous schemes. In the near future, we will implement more types of application programs to verify our approach. Also, automatic transforming legacy MPI programs to performance-based ones will be investigated.

REFERENCES

1. I. Foster, "The grid: A new infrastructure for 21st century science," *Physics Today*, Vol. 55, 2002, pp. 42-47.
2. I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of Supercomputer Applications and High Performance Computing*, Vol. 11, 1997, pp. 115-128.
3. C. Boeres, *et al.*, "An EasyGrid portal for scheduling system-aware applications on computational grids," *Concurrency and Computation: Practice and Experience*, Vol. 18, 2006, pp. 553-566.
4. C. Boeres and V. E. F. Rebello, "EasyGrid: Towards a framework for the automatic grid enabling of legacy MPI applications," *Concurrency and Computation: Practice and Experience*, Vol. 16, 2004, pp. 425-432.

5. A. P. Nascimento, *et al.*, "Distributed and dynamic self-scheduling of parallel MPI grid applications," *Concurrency and Computation: Practice and Experience*, Vol. 19, 2006, pp. 1955-1974.
6. O. Beaumont, *et al.*, "Scheduling divisible loads on star and tree networks: Results and open problems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, 2005, pp. 207-218.
7. D. Maciej and L. Marcin, "Multi-installment divisible load processing in heterogeneous systems with limited memory," *Parallel Processing and Applied Mathematics*, LNCS 3911, 2006, pp. 847-854.
8. D. Maciej and L. Marcin, "On optimum multi-installment divisible load processing in heterogeneous distributed systems," *Euro-Par 2005 Parallel Processing*, LNCS 3648, 2005, pp. 231-240.
9. C. Kruskal and A. Weiss, "Allocating independent subtaskson parallel processors," *IEEE Transactions on Software Engineering*, Vol. 11, 1984, pp. 1001-1016.
10. C. D. Polychronopoulos and D. J Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers," *IEEE Transactions on Computers*, Vol. 36, 1987, pp. 1425-1439.
11. H. S. Flynn, S. Edith, and E. F. Lawrence, "Factoring: A method for scheduling parallel loops," *Communications of the ACM*, Vol. 35, 1992, pp. 90-101.
12. 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.
13. 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, 2004, pp. 263-273.
14. C. T. Yang, K. W. Cheng, and K. C. Li, "An enhanced parallel loop self-scheduling scheme for cluster environments," *The Journal of Supercomputing*, Vol. 34, 2005, pp. 315-335.
15. W. C. Shih, C. T. Yang, and S. S. Tseng, "A performance-based parallel loop scheduling on grid environments," *The Journal of Supercomputing*, Vol. 41, 2007, pp. 247-267.
16. C. T. Yang, W. C. Shih, and S. S. Tseng, "Dynamic partitioning of loop iterations on heterogeneous PC clusters," *The Journal of Supercomputing*, Vol. 44, 2008, pp. 1-23.
17. The TIGER Grid, 2006, http://gamma2.hpc.csie.thu.edu.tw/ganglia/.
18. The Globus Project, 2004, http://www.globus.org/.
19. MPICH-G2, 2004, http://www.hpclab.niu.edu/mpi/.
20. J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, San Francisco, 2001.
21. R. Agrawal and J. C. Shafer, "Parallel mining of association rules," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, 1996, pp. 962-969.
22. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 487-499.
23. M. J. Zaki, "Parallel and distributed association mining: A survey," *IEEE Concurrency*, Vol. 7, 1999, pp. 14-25.
24. B. B. Mandelbrot, *Fractal Geometry of Nature*, W. H. Freeman, New York, 1988.

25. J. Herrera, *et al*., "Loosely-coupled loop scheduling in computational grids," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, 2006, pp. 6.
26. S. Penmatsa, *et al*., "Implementation of distributed loop scheduling schemes on the TeraGrid," in *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1-8.
27. M. Cannataro, *et al*., "Distributed data mining on grids: services, tools, and applications," *IEEE Transactions on Systems*, *Man*, *and Cybernetics – Part B*, Vol. 34, 2004, pp. 2451-2465.
28. V. Fiolet, *et al*., "Optimal grid exploitation algorithms for data mining," in *Proceedings of the 5th International Symposium on Parallel and Distributed Computing*, 2006, pp. 246-252.
29. W. S. Jiang and J. H. Yu, "Distributed data mining on the grid," in *Proceedings of the 4th International Conference on Machine Learning and Cybernetics*, 2005, pp. 2010-2014.

Wen-Chung Shih (時文中) received the Ph.D. degree in Computer Science from National Chiao Tung University in 2008. Since 2004, he has worked as a librarian in National Chi Nan University library, Taiwan. In August 2008, he joined the faculty of the Department of Information Science and Applications at Asia University, where he is currently an assistant professor. His research interests include e-learning, ubiquitous learning, grid computing and expert systems.

Chao-Tung Yang (楊朝棟) is a professor of Computer Science and Information Engineering at Tunghai University in Taiwan. He received a B.S. degree in Computer Science and Information Engineering from Tunghai University, Taichung, Taiwan, in 1990, and the M.S. degree in Computer and Information Science from National Chiao Tung University, Hsinchu, Taiwan, in 1992. He received the Ph.D. degree in Computer and Information Science from National Chiao Tung University in July 1996. He won the 1996 Acer Dragon Award for an outstanding Ph.D. dissertation. He has worked as an associate researcher for ground operations in the ROCSAT Ground System Section (RGS) of the National Space Program Office (NSPO) in Hsinchu Science-based Industrial Park since 1996. In August 2001, he joined the faculty of the Department of Computer Science and Information Engineering at Tunghai University. He got the excellent research award by Tunghai University in 2007. His researches have been sponsored by Taiwan agencies National Science Council (NSC), National Center for High Performance Computing (NCHC), and Ministry of Education. His present research interests are in grid and cluster computing, parallel and high-performance computing, and internet-based applications. He is both a member of the IEEE Computer Society and ACM.

Shian-Shyong Tseng (曾憲雄) received the Ph.D. degree in Computer Engineering from the National Chiao Tung University in 1984. Since August 1983, he has been on the faculty of the Department of Computer and Information Science at National Chiao Tung University, and is currently a Professor there. From 1988 to 1991, he was the Director of the Computer Center at National Chiao Tung University. From 1991 to 1992 and 1996 to 1998, he acted as the Chairman of Department of Computer and Information Science. From 1992 to 1996, he was the Director of the Computer Center at Ministry of Education and the Chairman of Taiwan Academic Network (TANet) management committee. From 1999 to 2003, he has participated in the National Telecommunication Project and acted as the Chairman of the Network Planning Committee, National Broadband Experimental Network (NBEN). From 2003 to 2006, he has acted as the principal investigator of the Taiwan SIP/ENUM trial project and the Chairman of the SIP/ENUM Forum Taiwan. In Dec. 1999, he founded Taiwan Network Information Center (TWNIC) and was the Chairman of the board of directors of TWNIC from 1999 to 2005. Since August 2005, he is the Dean of the College of Computer Science, Asia University. He is also the Director of the e-learning and application research center at National Chiao Tung University. His current research interests include expert systems, data mining, computer-assisted learning, and Internet-based applications. He has published more than 100 journal papers.