Performance Study of Parallel Programming on Cloud Computing Environments Using MapReduce

Wen-Chung Shih, Shian-Shyong Tseng
Department of Information Science and Applications
Asia University
Taichung, 41354, Taiwan
{wjshih, sstseng}@asia.edu.tw

Chao-Tung Yang*
Department of Computer Science and Information Engineering, Tunghai University
Taichung, 40704, Taiwan
ctyang@thu.edu.tw

Abstract—**Divisible load applications have such a rich source of parallelism that their parallelization can significantly reduce their total completion time on cloud computing environments. However, it is a challenge for cloud users, probably scientists and engineers, to develop their applications which can exploit the computing power of the cloud. Using MapReduce, novice cloud programmers can easily develop a high performance cloud application. To examine the performance of programs developed by this approach, we apply this pattern to implement three kinds of applications and conduct experiments on our cloud test-bed. Experimental results show that MapReduce programming is suitable for regular workload applications.**

***Keywords-** high performance computing; parallel programming; cloud computing; MapReduce; Hadoop*

I. INTRODUCTION

In recent years, integration of inexpensive commodity computers, such as computing clusters and grids, have become promising alternatives to traditional multiprocessors [1, 2]. Among these, cloud computing has emerged as the next-generation high performance computing platform. Basically, cloud platforms are distributed systems which share resources through the form of internet services. On the one hand, users can access more computing resources through cloud technologies without knowing low-level details. On the other hand, cloud environments require effective management to operate in an efficient way. Moreover, the heterogeneity and dynamic changing of the cloud environment make it different from conventional parallel and distributed computing systems, such as multiprocessors and computing clusters. Therefore, it is a challenge to utilize the cloud efficiently.

Applications with divisible loads are a rich source of parallelism [30]. 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 cloud 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 cloud status information and to appropriately distribute workload within a program to heterogeneous working nodes.

Our idea is to provide programmers with a parallel programming pattern, which takes care of details related to cloud infrastructure. All the programmers need to do is to fill in the pattern 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 pattern algorithm, which serves as a template for programmers to develop a parallel program on cloud platforms. To verify this approach, we have implemented this pattern using Hadoop MapReduce [31, 32] and applied this pattern to three types of applications, Matrix Multiplication, Association Rule Mining and Mandelbrot Set Computation. Experimental results on a cloud test-bed show that programs developed by this approach can exploit the computing power of the cloud.

The primary advantage of this approach is that a programmer can easily develop high performance programs to execute on cloud environments. The high performance results from two features of this pattern. 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 cloud. The pattern acquires cloud 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 pattern for programmers to develop high-quality parallel applications with ease. Programs developed by this approach can utilize cloud information to adaptively distribute workloads within a program. Second, we apply this pattern to three kinds of divisible load applications on our cloud test-bed. Consequently, experimental results show the obvious effectiveness of our approach. Note that this work aims at a general pattern of workload distribution, instead of proposing a new loop scheduling scheme or a novel data mining algorithm.

II. RELATED WORK

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

*Corresponding author

schemes.

A. *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 cloud platforms, 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 cloud environments. Since cloud environments are dynamically changing and heterogeneous, the problem is obviously different from the traditional DLT problem.

B. *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.

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. In [30], the idea of performance-based scheduling was extended to design a performance-based skeleton for developing high performance applications on grid platforms. This work differs from the above-mentioned researches in that the computing environment is focused on cloud platforms.

III. Approach

In this section, the MapReduce model is introduced first. Then, the concepts of performance ratio and static-workload ratio are reviewed. Finally, we present the programming pattern for the performance-based cloud computing.

A. MapReduce Model

The MapReduce programming model can be used to process large-scale data sets in cloud environments. It consists of three types of nodes: Master, Mappers and Reducer, as shown in Figure 1. The Master dispatches sub-jobs to a set of Mappers. After these Mappers complete their assigned jobs, the results are merged by the Reducer.

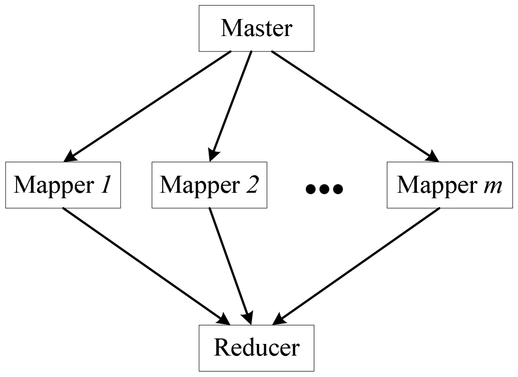


Figure 1. Overview of the MapReduce model

Hadoop [31] implements the MapReduce model, which provides a high-level view to parallel programmers. With the help of Hadoop, programmers can focus on high-level operations. In the MapReduce model, the Mappers get sub-jobs in the form of <key, value>. Taking a large data set as example, a sub-job can be represented by < *key*, *value* >, where *key* is the filename of the data subset, and *value* is the content of the data subset.

B. 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 pattern algorithm uses a performance function to model the heterogeneous performance of the dynamic cloud 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*, as shown in (1):

$$PF_j\,(V_1, V_2, \ldots, V_m) \qquad (1)$$

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 Cloud resource monitoring tool, Hadoop, to acquire the values of attributes for all slaves. The PF for node *j* is defined as (2).

$$PF_j = \frac{CS_j / CL_j}{\sum_{\forall node_i \in N} CS_i / CL_i} \qquad (2)$$

where

- N is the set of all available cloud 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 monitoring 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 monitoring tool.

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.

C. Determination of Static-Workload Ratio (SWR)

Another important factor to be estimated is the variation degree among all units of workloads. The concept of Static-Workload Ratio (SWR) was previously defined in [30]. 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 Figure 2. Obviously, a distribution scheme which does not consider the effect of irregular workload could not estimate PR accurately.

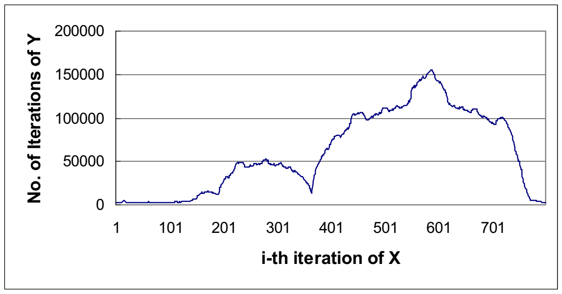


Figure 2. The Mandelbrot Set on [-1.8, 0.5] to [-1.2, 1.2]

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 (3).

$$SWR_i = \frac{min_i}{MAX_i} \qquad (3)$$

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.

D. Programming Pattern

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

Module Map

```
Initialization

/* Stage 1: Gathering the information */
  collect the following information:
        –  CPU_Loading
        –  CPU_Clock_Speed
  collect the execution time of 5 sampled
iterations

/* Stage 2: Calculate 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 Map
```

Module Reduce

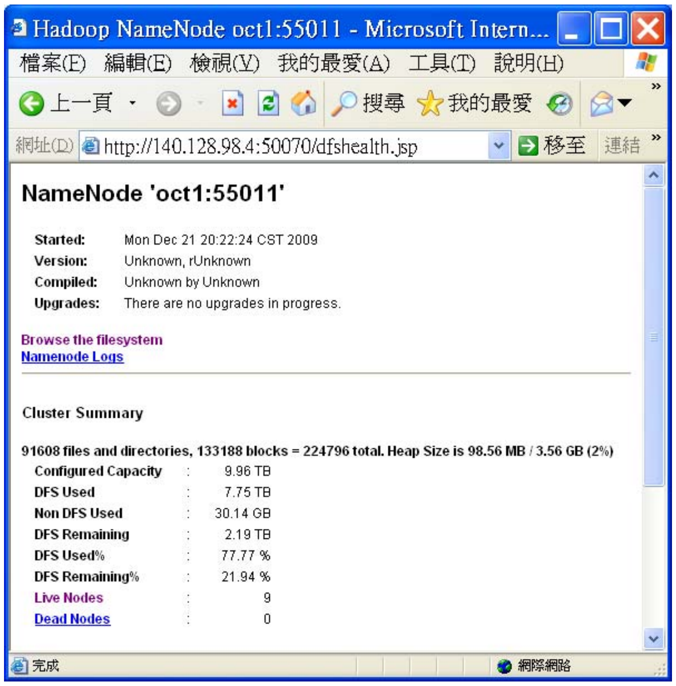
```
Initialization
While (a chunk of workload arrives) {
   receive the chunk of workload
     Compute on this chunk
   Send the result to the Master
}
Finalization
END Reduce
```

IV. Experimental Results

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

A. Cloud Test-bed

A Cloud computing test-bed has been built by the High Performance Computing Lab. of Tunghai University, Taiwan [33], using Hadoop. The summary of the nodes is shown in Figure 3. Figure 4 shows the real-time status of the cloud test-bed acquired by the monitoring tool.


Hadoop NameNode oct1:55011 - Microsoft Intern...
檔案(F) 編輯(E) 檢視(V) 我的最愛(A) 工具(T) 說明(H)
上一頁 搜尋 我的最愛
網址(D) http://140.128.98.4:50070/dfshealth.jsp 移至 連結
完成 網際網路

NameNode 'oct1:55011'

Started:	Mon Dec 21 20:22:24 CST 2009
Version:	Unknown, rUnknown
Compiled:	Unknown by Unknown
Upgrades:	There are no upgrades in progress.

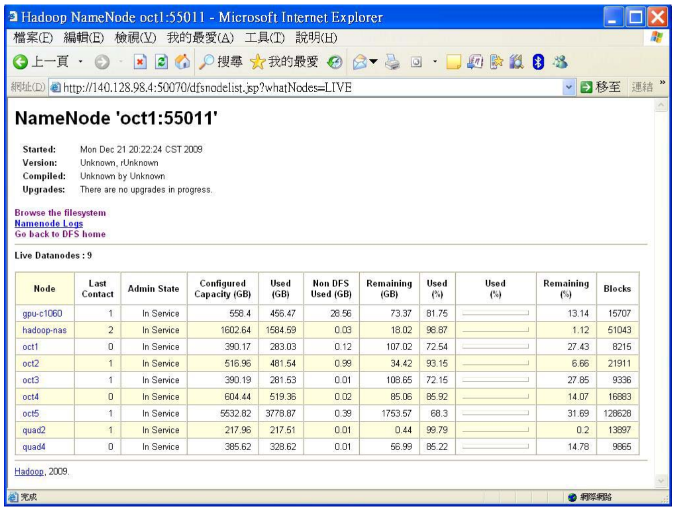
Browse the filesystem
Namenode Logs

Cluster Summary

91608 files and directories, 133188 blocks = 224796 total. Heap Size is 98.56 MB / 3.56 GB (2%)

Configured Capacity	:	9.96 TB
DFS Used	:	7.75 TB
Non DFS Used	:	30.14 GB
DFS Remaining	:	2.19 TB
DFS Used%	:	77.77 %
DFS Remaining%	:	21.94 %
Live Nodes	:	9
Dead Nodes	:	0

Figure 3. The node summary of the cloud test-bed



Hadoop NameNode oct1:55011 - Microsoft Internet Explorer

http://140.128.98.4:50070/dfsnodelist.jsp?whatNodes=LIVE

NameNode 'oct1:55011'

Started:	Mon Dec 21 20:22:24 CST 2009
Version:	Unknown, rUnknown
Compiled:	Unknown by Unknown
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs
Go back to DFS home

Live Datanodes : 9

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
gpu-c1060	1	In Service	558.4	456.47	28.56	73.37	81.75		13.14	15707
hadoop-nas	2	In Service	1602.64	1584.59	0.03	18.02	98.87		1.12	51043
oct1	0	In Service	390.17	283.03	0.12	107.02	72.54		27.43	8215
oct2	1	In Service	516.96	481.54	0.99	34.42	93.15		6.66	21911
oct3	1	In Service	390.19	281.53	0.01	108.65	72.15		27.85	9336
oct4	0	In Service	604.44	519.36	0.02	85.06	85.92		14.07	16883
oct5	1	In Service	5532.82	3778.87	0.39	1753.57	68.3		31.69	128628
quad2	1	In Service	217.96	217.51	0.01	0.44	99.79		0.2	13897
quad4	0	In Service	385.62	328.62	0.01	56.99	85.22		14.78	9865

Hadoop, 2009.

Figure 4. The snapshot of the monitoring tool on the TIGER Cloud

In this study, we have implemented several scheduling schemes for the purpose of evaluation. 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 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.

B. *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.

First, we want to compare the proposed PWD scheme with previous schemes with respect to the execution time. Figure 5 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.

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 cloud 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.

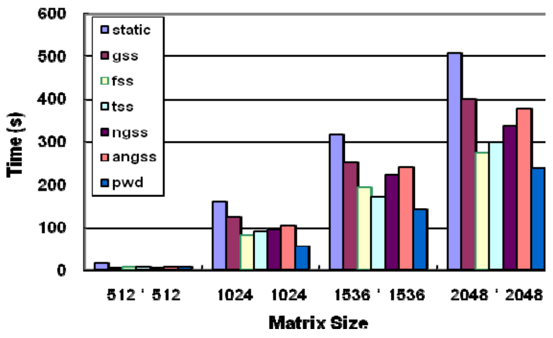


Figure 5. Execution time for Matrix multiplication with different input sizes

C. *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 pattern 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 I.

TABLE I. 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

Figure 6 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 pattern 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 cloud 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 cloud 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.

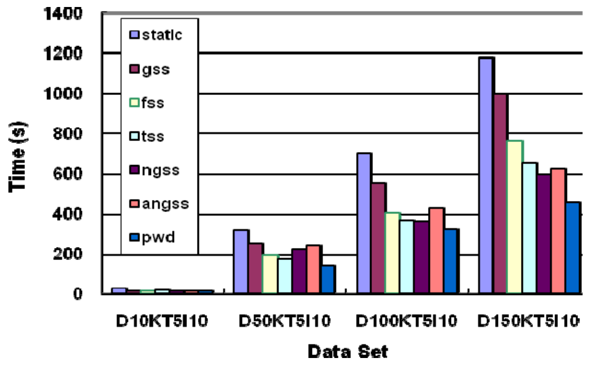


Figure 6. Performance of data partition schemes for different datasets

D. *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 pattern. 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.

In the following experiment, we want to compare the execution time of previous schemes with the implemented program. Figure 7 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.

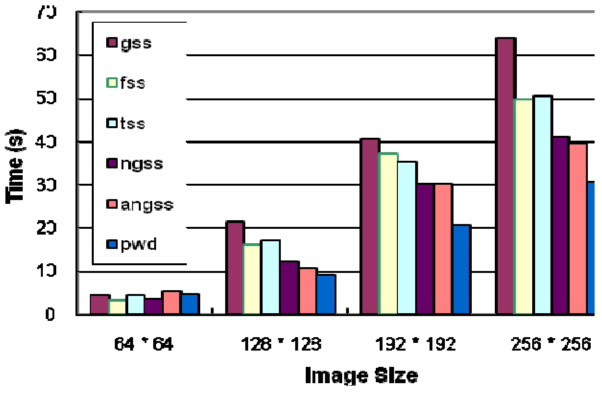


Figure 7. Execution time for Mandelbrot Set Computation with different input sizes

E. Discussion

In this section, several issues are discussed to clarify the proposed approach. In general, task scheduling in cloud systems mainly focuses on fine grain parallelism, under the consideration of the system heterogeneity and the message-passing communication. However, one goal of cloud computing is to exploit potential parallelism in internet-scale cloud environments. In addition to coarse grain parallelism, we think that it is beneficial to exploit fine grain parallelism in cloud systems. The first reason is to improve utilization. The proposed approach provides a mechanism for programmers to efficiently utilize the idle resources located in cloud 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 cloud 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 cloud 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 cloud programming pattern, 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 pattern abstracts our experiences in programming master-slave applications for cloud environments, which is a difficult task for novice programmers. Nevertheless, with the pattern, all a programmer need to do is just to fill the application-specific program codes into the pattern. If a programmer can code a sequential program, then it is straightforward to transform it to a cloud application. To extend the pattern 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.

V. Conclusions

We have proposed a programming pattern for programmers to easily develop high performance applications on dynamic and heterogeneous cloud environments. This pattern 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 cloud 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, application of performance-based programming to e-learning will be investigated.

Acknowledgment

The work is supported in part by National Science Council, Taiwan R.O.C., under grant no. NSC97-2511-S-468-003, NSC98-2511-S-468-002, NSC97-2511-S-009-001-MY3 and NSC98-2511-S-468-004-MY3.

References

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

[2] Foster, I. and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications and High Performance Computing, 1997. 11(2): p. 115-128.

[3] Boeres, C., et al., An EasyGrid portal for scheduling system-aware applications on computational Grids. Concurrency and Computation: Practice and Experience, 2006. 18(6): p. 553-566.

[4] Boeres, C. and V.E.F. Rebello, EasyGrid: towards a framework for the automatic Grid enabling of legacy MPI applications. Concurrency and Computation: Practice and Experience, 2004. 16(5): p. 425-432.

[5] Nascimento, A.P., et al., Distributed and dynamic self-scheduling of parallel MPI Grid applications. Concurrency and Computation: Practice and Experience, 2006. in press.

[6] Beaumont, O., et al., Scheduling divisible loads on star and tree networks: results and open problems. Parallel and Distributed Systems, IEEE Transactions on, 2005. 16(3): p. 207-218.

[7] Maciej, D. and L. Marcin, Multi-installment Divisible Load Processing in Heterogeneous Systems with Limited Memory. : Parallel Processing and Applied Mathematics. 2006. 847-854.

[8] Maciej, D. and L. Marcin, On Optimum Multi-installment Divisible Load Processing in Heterogeneous Distributed Systems. : Euro-Par 2005 Parallel Processing. 2005. 231-240.

[9] Kruskal, C. and A. Weiss, Allocating independent subtaskson parallel processors. IEEE Transactions on Software Engineering, 1984. 11: p. 1001-1016.

[10] Polychronopoulos, C.D. and D.J. Kuck, Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. 1987, IEEE Computer Society. p. 1425-1439.

[11] Susan Flynn, H., S. Edith, and E.F. Lawrence, Factoring: a method for scheduling parallel loops. 1992, ACM Press. p. 90-101.

[12] Tzen, T.H. and L.M. Ni, Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers. IEEE Transactions on Parallel and Distributed Systems, 1993. 4(1): p. 87-98.

[13] Yang, C.T. and S.C. Chang, A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters. Journal of Information Science and Engineering, 2004. 20(2): p. 263-273.

[14] Yang, C.T., K.W. Cheng, and K.C. Li, An Efficient Parallel Loop Self-Scheduling Scheme for Cluster Environments. Journal of Supercomputing, 2005. 34: p. 315-335.

[15] Shih, W.-C., C.-T. Yang, and S.-S. Tseng, A performance-based parallel loop scheduling on grid environments. The Journal of Supercomputing, 2007. 41(3): p. 247-267.

[16] Yang, C.-T., W.-C. Shih, and S.-S. Tseng, Dynamic Partitioning of Loop Iterations on Heterogeneous PC Clusters. to appear in The Journal of Supercomputing, 2008.

[17] THU. The TIGER Grid. 2006 [cited 2006; Available from: http://gamma2.hpc.csie.thu.edu.tw/ganglia/.

[18] Globus. The Globus Project. 2004 [cited 2006; Available from: http://www.globus.org/.

[19] MPICH. MPICH-G2. 2004 [cited 2006; Available from: http://www.hpclab.niu.edu/mpi/.

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

[21] Agrawal, R. and J.C. Shafer, Parallel Mining of Association Rules. IEEE Transactions on Knowledge and Data Engineering, 1996. 8(6): p. 962-969.

[22] Agrawal, R. and R. Srikant. Fast algorithms for Mining Association Rules. in Proc. 20th Very Large Data Bases Conf. 1994.

[23] Zaki, M.J., Parallel and Distributed Association Mining: A Survey. IEEE Concurrency, 1999. 7(4): p. 14-25.

[24] Mandelbrot, B.B., Fractal Geometry of Nature. 1988, New York: W. H. Freeman.

[25] Herrera, J., et al. Loosely-coupled loop scheduling in computational grids. in The 20th International Parallel and Distributed Processing Symposium (IPDPS 2006). 2006.

[26] Penmatsa, S., et al. Implementation of Distributed Loop Scheduling Schemes on the TeraGrid. in IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007). 2007.

[27] Cannataro, M., et al., Distributed data mining on grids: services, tools, and applications. Systems, Man, and Cybernetics, Part B, IEEE Transactions on, 2004. 34(6): p. 2451-2465.

[28] Fiolet, V., et al. Optimal Grid Exploitation Algorithms for Data Mining. in The Fifth International Symposium on Parallel and Distributed Computing (ISPDC '06). 2006.

[29] Jiang, W.-S. and J.-H. Yu. Distributed data mining on the grid. in Proceedings of 2005 International Conference on Machine Learning and Cybernetics 2005.

[30] Shih, W.C., Yang C.T. and Tseng S.S., "Using a Performance-based Skeleton to Implement Divisible Load Applications on Grid Computing Environments," *Journal of Information Science and Engineering (JISE), 25(1),* pp. 59-81, 2009.

[31] Apache, "Hadoop", *http://lucene.apache.org/hadoop/*, 2006.

[32] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI'04: Proceedings of the 6th Symposium on Operating System Design and Implementation*. San Francisco, California, USA. USENIX Association, pp. 137-150. December 6-8, 2004.

[33] THU, "Cloud Testbed", http://140.128.98.4:50070/dfshealth.jsp, 2009.