View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Tamkang University Institutional Repository

Dynamically Iterative MapReduce

Wei-Tsong Lee, Ming-Zhi Wu, Hsin-Wen Wei, Fang-Yi Yu , Yu-Sun Lin*
Department of Electrical Engineering, TamKang University, Taiwan
Chung-Shan Institute of Science and Technology, Taiwan, R.O.C
hwwei@mail.tku.edu.tw

Abstract

MapReduce is a distributed and parallel computing model for data-intensive tasks with features of optimized scheduling, flexibility, high availability, and high manageability. MapReduce can work on various platforms; however, MapReduce is not suitable for iterative programs because the performance may be lowered by frequent disk I/O operations. In order to improve system performance and resource utilization, we propose a novel MapReduce framework named Dynamically Iterative MapReduce (DIMR) to reduce numbers of disk I/O operations and the consumption of network bandwidth by means of using dynamic task allocation and memory management mechanism. We show that DIMR is promising with detail discussions in this paper.

Keywords: *Dynamically Iterative MapReduce, K-Means, Particle Swarm Optimization (PSO), Genetic Algorithm (GA) , Simulated Annealing (SA).*

1 Introduction

Information technology industry has been changed by cloud computing technology [1]. Programmers can just focus on developing software to improve service quality and application performance instead of upgrading their hardware. By renting computing resources from cloud service providers [2], IT industry does not need to purchase or construct infrastructure. Cloud computing technology also reduces manpower and the cost of upgrading software and hardware. Cloud computing includes various services stored in remote environments [3], and all users only have to know is to choose the service and use it via networks. To fulfill the requirements of users, cloud computing techniques are needed to carry out a large number of computations with highly parallel and distributed processing.

MapReduce [4] framework is one of well-known cloud computing techniques that can help the user to complete the task efficiently, especially in dealing with a large number of batch files. The hardware of nodes used in MapReduce computation does not need to be as good as high-performance computing computer, but just low-cost, low-performance computers like what we use in daily life. MapReduce runtime system automatically divide input file into the number of copies, and automatically deal these data with scheduling, load balancing, fault tolerance, and managing, and so on. MapReduce allows the user to finish their job in parallel without any knowledge of the distribution of their work. Users only need to write Map and Reduce function, and MapReduce Execution System function, and then the job will be completed according to the function written by the users.

Many optimization solvers, such as K-Means [5][20], Particle Swarm Optimization (PSO) [6][19], Genetic Algorithm (GA) [7], Simulated Annealing (SA) [8], and image processing application [18], need to process large amounts of information, and numerous iterations as well as high-density computing. In addition, many analysis and statistical results of the Web content [16], the commercial transaction information, and Bioinformatic data are needed for various applications, which require a lot of time for processing large amounts of data. However, single operational unit does not have the ability to satisfy the requirement for processing large amounts of data. Therefore, MapReduce will be the solution for the applications that need large amount of data computation. Using MapReduce can decrease the overall processing time, increase the efficiency, reliability, and enhance integrity by fault tolerance mechanism.

While using MapReduce can solve the problem of a single computing unit, but the traditional computing architecture of MapReduce will make the performance of some applications like optimization algorithms degrade. Since optimization algorithms need to go through numerous iterations and require high-density computing to find the final answer, the large amount of data migration and pass overhead will decline system performance. In addition, the design of traditional MapReduce aims to handle large and parallelable tasks, not designed for handling tasks with multiple iterations of data. Hence, to improve the performance of using MapReduce to handle a program with iterative structure will be an urgent problem that needs to be solved. However, the computing nodes in the MapReduce operational environment do not have the same computing capability, i.e., CPU computing power, storage space,

*Corresponding author: Hsin-Wen Wei; E-mail: hwwei@mail.tku.edu.tw

and network bandwidth, are not same on every node. Using a heterogeneous environment increases the difficulty of the assignment and affects the overall operational efficiency and reliability. Therefore, this paper proposes a new type of MapReduce architecture called Dynamically Iterative MapReduce (DIMR), and it aims to reduce the overhead of passing data between large numbers of iterations and to increase the performance of applications on the MapReduce runtime system using the heterogeneous environment.

The remainder of this paper is organized as follows. We first give an overview of the well-known MapReduce runtimes in Section2. Previous works are presented and discussed in Section 3. Section 4 presents the DIMR architecture and workflow. The experimental results are shown in Section 5 and the paper is concluded in Section 6.

2 Background

2.1 Hadoop MapReduce

The MapReduce architecture in Hadoop [9] is shown in Fig. 1. MapReduce is composed by the one Master as well as a number of Map blocks and Reduce blocks. MapReduce works through the several stages of operation to complete the data processing. First, Master Node will receive the command of the user to start a MapReduce program and have knowledge of the position and the size of the input file in HDFS [10]. Then, Master Node will assign the idle Worker Node to execute Map function or Reduce function. Master Node will distribute input file evenly according to the number of Map functions, and the file blocks assigned to Map function are called split. Therefore, the number of splits generated by Master Node is equal to the number of Map functions. Note, Master Node does not transfer the input file directly to the Map Node, but inform Map Node about the location and the size of needed file. If Master Node has distributed tasks to all available Worker Node, then Master Node constantly monitor the states of all Map function for finding the idle Worker Node. Master Node will continuously assign the task to idle Worker Node or a Worker Node that finish its work until all tasks are completed. The Map function in Map Node will receive the location and size of split from Master Node and issue the requirement over HTTP to ask for data in HDFS. Map function is initiative to retrieve data from HDFS; nevertheless, each Map function in this step is carried out simultaneously and in accordance with user-defined way to deal with these data.

The Reduce function in Reduce Node will receive the message sent by Master Node. The Reduce function will know the location and size of the intermediate data by the message, and issue the requirement over HTTP to ask for the intermediate data which is assigned by Map Node. The time required for Reduce function to receive the message from Master Node determined by the processing speed of Map Node, so Reduce function is non-synchronized receiving and processing intermediate data. Next step, Reduce function will simplify these data according to the user definition. In the end, Reduce function will finish the computation, and produce an output file and store it into HDFS. Then, Worker Node will notify the Master Node that task is completed and waiting for the next task. When MapReduce components complete all the tasks, it means the job has finished, and then they will continue to the next work.

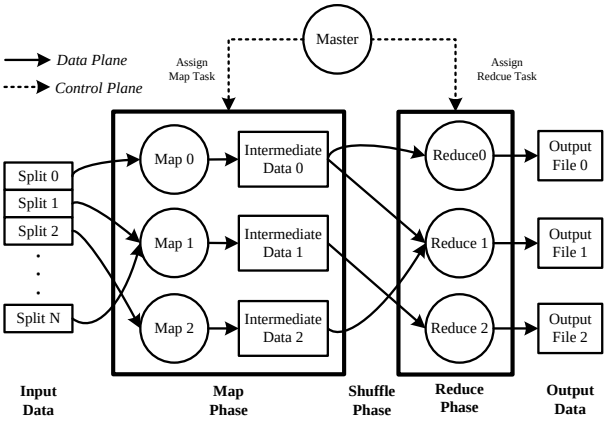

Figure 1 Hadoop MapReduce Architecture

2.2 Twister

Twister [11], which is a MapReduce runtime, aims to solve the overhead caused by hierarchical MapReduce applications, since hierarchical MapReduce must constantly and repeatedly to create a new Map function and Reduce function. The architecture of Twister as shown in Fig. 2 makes it has better performance than other MapReduce framework in dealing with repeated (iterative) calculation. When Twister receives the computing request from users, it will distribute all tasks and execute Map function and Reduce function. In the next step, Twister Worker Node will start to execute the tasks. Map function in the Twister will read the local disks or remote disks to find data which is needed to be computed. After Twister receives the data, the Map function starts the computation and produce intermediate data as an output file. Intermediate data in Twister is distributed to memory of each Worker Node. When the Map function in Twister finishes the job, intermediate data will be directly transferred to appropriate Reduce function.

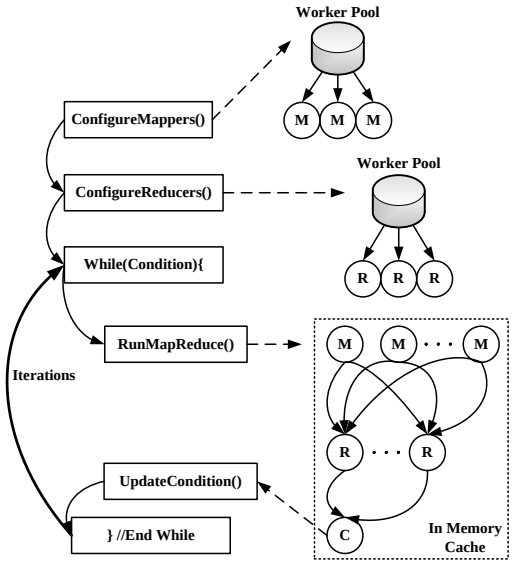

Figure 2 Twister MapReduce Architecture [11]

The Reduce function in Twister will store intermediate data in the memory cache until Reduce function execution is completed. Note that, these intermediate data can also be stored in the local disk. After the Reduce function in Twister completes the task, and then the output file will be sent to the Combiner. The main job of Combiner in the Twister is to combine the result generated by Reduce function, and produce a new output file to the Map function. When the Combiner in Twister finishes the combination of the output files that Reduce functions completed, it will transfer the result to next Map function to start next MapReduce iteration.

Twister is different from other MapReduce framework, it uses a publish/subscribe messaging architecture for communication and data transmission. In addition, the intermediate data generated by Map function in Twister will be transferred directly to the Reduce function and the output file generated by Reduce function in Twister will be directly transferred to the Combiner. Therefore, Twister architecture effectively reduces overhead of storing data into storage and passing control messages. In other words, Twister saves the time spent on I/O.

3 Related Work

To increase the performance of handling optimization problems, there are many optimization algorithms implemented in parallel way and computed in distributed environment. There are some important MapReduce systems and methods address the data migration issues in optimization problems and network I/O bottleneck [17]. Although these methods are faster than that using single-node operation, they are not efficient solutions in using the whole system resources. In [6], the authors proposed MRPSO to find the best answer of PSO problem by writing a simple Map/Reduce function and utilize the Map function to find the local optimum and then call the Reduce function to find the global optimum. The operating environment used in [6] is Hadoop basically, so the data completed by Reduce function is written to HDFS. Then, the Map function will read the previous generated file from HDFS and do the same job as above. It takes a lot of time for Map function and Reduce function to access data from HDFS, and therefore degrades the system performance.

A MapReduce operation method called MRPGA is proposed in [12], which aims to complete the calculation of the GA by parallel and distributed computing. The MRPGA finds the best answer by two-phase Reduce. In [12], the Reduce function of the second phase must withstand a lot of input pressure, and it will inevitably result in inferior system performance. Moreover, its operating environment is like Hadoop, so the output files must be stored via HDFS, and the Map function must access the input files from HDFS. This will cause system I/O busy, and also decrease the performance.

The proposed method in [13] combined two optimization algorithms PSO and GA to find the best answer quickly and accurately. The solution in [13] utilizes the Map function to execute PSO part and the Reduce function to execute GA mutation and cross-over. Although the solution in [13] makes the computing speed up, but Hadoop is not suitable iterative program originally, so data is not properly utilized.

An algorithm called Adaptive Disk I/O Scheduling for MapReduce is proposed in [14], which adjust I/O computing process to improve the overall performance and reduce the I/O bottleneck. In [14], the authors present the Task Scheduler architecture, which optimizes various MapReduce applications and manage the I/O process, as well as the resources required for Map functions and Reduce functions. The experimental results show that the performance is improved more than 25%. Nevertheless, the architecture is not suitable for applications with large numbers of iterations, which will lead to performance degradation.

Though, above solutions address the problems caused by iterative programs, they does not utilize the resource at each node effectively in fact. Therefore, we proposed a novel MapReduce runtime, called Dynamically Iterative MapReduce (DIMR), to solve

the overhead caused by multiple applications with iterative structures.

4 Dynamically Iterative MapReduce

4.1 DIMR Overview

This paper presents a MapReduce framework that can dynamically determine the location of output data to reduce most of execution time of iterative programs, is called Dynamically Iterative MapReduce (DIMR). DIMR can reduce the bottlenecks of I/O operations in iterative MapReduce applications which require a large amount of iterative I/O operations, such as the optimization algorithm, recursive, heuristic, and so on. DIMR is shown as Fig. 3, which is composed of a Master Node and a resource pool of multiple Work Nodes, and file system. The main controller of DIMR is in the Master Node, which manages the resources, tasks distribution, error recovery, and the most important is to collect the usage of each node for performance evaluation. The Master of DIMR will follow the settings to allot tasks to Work Nodes to execute Map function and Reduce function. Master will instruct Work Node to execute first iteration, and collect the information of system performance; finally to allot resources and tasks by the performance diagnosis. The performance of system is evaluated in each iteration and then Master will decide the operation in next iteration according to the tasks efficiency.

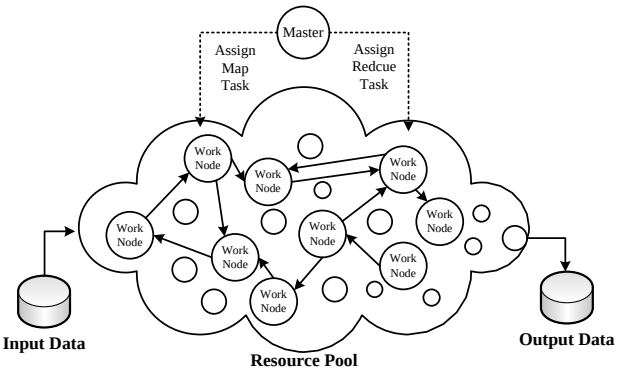

Figure 3 DIMR Overview

Different from the conventional MapReduce, in DIMR, Work Node acquires unprocessed data from file system and places the results into the Memory, after it finish the computation, it will then transfer the result directly to the next Work Node. This way can effectively reduce the time of writing data into the local disk and read data from the remote disk. Therefore, DIMR can achieve a better performance by reducing the overhead of network I/O through controlling the assignment of Map function and Reduce function. Overall, DIMR has excellent performance in processing and is efficient in resources use, it not only can reduce processing time of a job which needs a lot of iterations, but also can efficiently reduce the I/O overhead.

4.2 DIMR Working Principle

Figure 4 shows the DIMR working principle, consisting of 6 main stages and each stage performs different work. Master assigns those Work Nodes to handle Map function as a Map Node or handle Reduce function as a Reduce Node, and receive the message from the Work Nodes regularly and assign tasks to those nodes. The Map Node and Reduce Node capture files, transfer files, or handle files according to the instruction from Master, and report progress and the information about execution performance to Master Node.

In stage one, the Map function in Map Node will follow the instructions of the Master to capture the input file from file system, called Copy Phase, which shows in Fig. 4(1). Then, Map Node will call the Map function process tasks according to the user-written Map function, called Map Phase, which shows in Fig. 4(2). In stage three, the Map Node will generate intermediate data after finish the assigned task and store those intermediate data in memory and send the progress report to Master including data size and number of keys, which shows in Fig. 4(3). Then, Master instructs Map Node to transfer those intermediate data to Reduce Node after evaluating overall performance of system. Reduce Node received data from the Map Node via reliable TCP protocol, which shows in Fig. 4(4) as Merge Phase. Reduce Node will wait for all intermediate data collection completed and call Reduce function to perform these intermediate data, which shows in Fig. 4(5) as Reduce Phase. Then, Reduce Node will send this intermediate data and progress report to Master including data size and number of keys, and information about execution performance.

At this point, Master has collected the complete information about execution performance in the first iteration, and decides resource arrangement in the next iteration after finish performance diagnosis. The Master will notify the Reduce Node to send the intermediate data to a specified Map Node as the input data of Map function, if the number of iterations does not meet the requirement. Otherwise, the final result will be written into the file system if the Master has completed the operation, which shows in Fig. 4(6) as Write Phase.

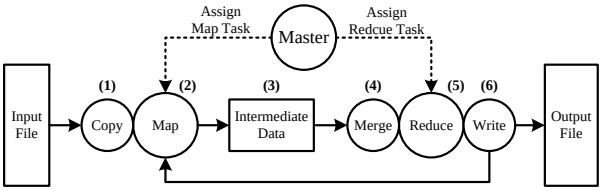

Figure 4 DIMR Working Principles

4.3 DIMR Components

Master Node controls and manages the entire the MapReduce operation environment as well as the allocation of resources in DIMR. As shown in Fig. 5, DIMR is composed of five components; there are Job Queue, Iteration Monitor, Node Manager, Task Manager and Scheduler respectively. The five important components cooperate with each other, and Master assign tasks and sources dynamically, to achieve the best utilization of computing resource and to finish the required job efficiently.

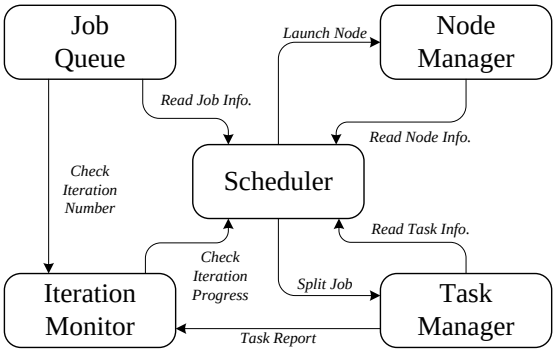

Figure 5 DIMR Components

4.3.1 Job Queue

The Job Queue stores parameters for computing operation and uncompleted jobs. Job Queue will obtain the Map function and Reduce function written by the users, including the number of nodes, the size of blocks, the location and size of input file, and the count of iterations. Next, Job Queue transfers the information to scheduler including the number and position of Map and Reduce nodes, and also transfers the count of iterations to Iteration Monitor. Job Queue traces the tasks' states constantly and check whether the setting parameters meet the requirements as predefined.

4.3.2 Iteration Monitor

The Iteration Monitor obtains the count of iterations, and traces the jobs' state. It will collect the information to inform Scheduler whether the works completed or not. Iteration Monitor gets the max count I from Job Queue and the job's state i_now from Task Manager. According to the Eq. (1), we can find that whether Schedule continues to distribute work or not, it is.

$$Iteration\ Index = \begin{cases} Final & i_{now} \geq I \\ Next & otherwise \end{cases} \quad (1)$$

4.3.3 Node Manager

Node Manager manages the resources of nodes and traces the number of resource entities assigned to each job. Node Manager will receive the information of tasks' utilization and transfer it to Scheduler to distribute the resources. At first, Node Manager will receive the massage about how many number of Map function and Reduce function should be opened from Scheduler. Next, Node Manager will assign Work Node to execute the Map function and Reduce function according to the indication of Scheduler. Then, Node Manager will obtain the information about computing efficiency of each node, and report it to Scheduler, so that Node Manager can distribute the resources according to the command of Scheduler to achieve the best performance.

When Node Manager assigns works to Work Node, it must follows the rules of Eq.(2) and Eq.(3). The total number of Map Node (N_m) , Reduce Node (N_r) as well as Map and Reduce Node (N_k) launched at the same time cannot be more than the number of available Node(N_available). However, to ensure the MapReduce keep working, the number of Map functions and Reduce functions must be more than two; it means that there are at least one Map function and one Reduce function in the system.

$$\sum_{m=0}^{M} N_m + \sum_{r=0}^{R} N_r + \sum_{k=0}^{K} N_k \leq N_{available} \quad (2)$$

$$\sum_{m=1}^{M} F_m + \sum_{r=1}^{R} F_r \geq 2 \quad (3)$$

4.3.4 Task Manager

Task Manager is in charge of managing and tracking the status of current task, if there is any error or delay, the Scheduler will be noticed. Task Manager will receive the schedule of each node and transmit the information to Scheduler after finishing the arrangement. Scheduler notifies Task Manager to manage input data and obtain the number of data segments and their sizes. The size of a data segment is defined in Eq.(2) to ensure the data segment received by each Map are in the same size. Next, Scheduler will assign tasks index and metadata to the specified Node.

$$Splits\ Size = \frac{Input\ File\ Size}{F_m} \quad (4)$$

4.3.5 Scheduler

When the task/node information is published by Task Manager and Node Manager, Scheduler will decide a way to execute next iteration by information that Task Manager and Node Manager provide. The

performance information that Scheduler produced at i-th iteration includes input file size (x_i), intermediate data size (y_i), Map execution time (Tm_i), Reduce execution time (Tr_i), merge time (Tg_i) and transmission time (Ts_i). As shown in Eq.(5), the time required to compute input data with size x_i at the i-th iteration is defined as $T_i(x_i)$ and the summation of time spent in each iteration for computing a job is defined in Eq.(6). The sum of time required for obtaining data from file system(Tm_{copy}), storing data which is generated by Reduce to file system(Tr_{write}) and processing computation in each iteration is defined in Eq.(7), which is the total time required for completing a Job. In order to minimize to total execution time as Eq.(8), Scheduler plays a huge role in assigning tasks and managing calculation resource.

$$T_i(x_i) = Tm_i + Tr_i + Tg_{i+} + Ts_i \quad (5)$$

$$\sum_{i=0}^{I} T_i(x_i) = \sum_{i=0}^{I}(Tm_i + Tr_i + Tg_{i+} + Ts_i \quad (6)$$

$$T = Tm_{copy} + Tr_{write} + \sum_{i=0}^{I} T_i(x_i) \quad (7)$$

$$T_{min} = min \sum_{i=0}^{I} T_i(x_i) \quad (8)$$

However, Scheduler has to readjust the next execution at every iteration to minimize the system execution time. In order to lower the network bandwidth consumption, Scheduler will merge the data with lower Key value to the Node (n) with higher Key value data in it. Moreover, the Map Node will be transformed into a Reduce Node, which handles the Reduce function on the same Node. With this algorithm, the time required for transmitting intermediate data over Internet has been reduced. As shown in Eq.(9), the total transmission time at i-th iteration Ts_i is equal to the maximum execution time at i-th iteration among all Nodes.

$$Ts_i = max\ Ts_i^n \text{ , } 0 \le n \le N \text{ , } 0 \le i \le I \quad (9)$$

Each Map Node and Reduce Node will get Key through Scheduler, which make Map function and Reduce function have better management of tasks with same Key and Value by reducing the iteration count. The problem is that the Node will be overloaded with numerous tasks carrying the same Key. Other Nodes will become idle until the overloaded Node finishes its job, and this will lower the system performance. MapReduce managing and storing data inside system memory, in order to obtain higher performance, however, Scheduler have to consider the relationship between system memory and size of task. To solve this problem, we have two steps in our system.

Step 1, Scheduler transforms a Node with highest number of Key to Function Node at next step, which collects and manages Keys with the same value. We define the Key that Reduce Node n managed at i-th iteration as $\gamma_1(key_i^n)$, and Intermediate Key produced by Map Node n at i-th iteration as $m_1(key_i^n)$, see Eq(10). On the contraty, $m_1(key_i^n)$ is the Key that Map Node n managed at i-th iteration, $\gamma_1(key_i^n)$ is the Intermediate Key produced by Reduce Node n at (i-1)th iteration, see Eq(11). At Step 2, we use Greedy algorithm to solve the problem of the remaining Key assigned, each Node should be able to perform better, and thus, improve the system performance. As shown in Eq(12), $\gamma_2(key_i^n)$ is the Key assigned to Reduce Node n at i-th iteration, which is also the amount of unassigned intermediate data at i-th iteration, and is limited by the maximum memory size of the Node. On the contrary, $m_2(key_i^n)$ is the Key assigned to Map Node *n* at i-th iteration, which is also the amount of unassigned intermediate data at (i-1)th iteration, and is limited by the maximum memory size of the Node, see Eq.(13). After these steps, the Key of unmanaged intermediate data received by each Node n are $\gamma(key_i^n)$ in Eq.(14) and $m(key_i^n)$ in Eq.(15).

$$\gamma_1(key_i^n) \subseteq m_1(key_i^n)\ \forall\ n, i : n \le N, i \le I \quad (10)$$

$$m_1(key_i^n) \subseteq \gamma_1(key_{i-1}^n)\ \forall\ n, i : n \le N, i \le I \quad (11)$$

$$\gamma_2(key_i^n) = \{\sum_{n=1}^{N} m_2(key_i^n) \setminus \sum_{n=1}^{N} \gamma_1(key_i^n)\ :\ \sum_{k=1}^{K} S_i^n \le MemorySize_i^n\} \quad (12)$$

$$m_2(key_i^n) = \{\sum_{n=1}^{N} \gamma_2(key_{i-1}^n) \setminus \sum_{n=1}^{N} m_1(key_i^n)\ :\ \sum_{k=1}^{K} S_i^n \le MemorySize_i^n\} \quad (13)$$

$$\gamma(key_i^n) = \gamma_1(key_i^n) + \gamma_2(key_i^n) \quad (14)$$

$$m(key_i^n) = m_1(key_i^n) + m_2(key_i^n) \quad (15)$$

5 Performance Analysis

5.1 Experimental Environment

In order to evaluate the performance of DIMR, we build a large cloud cluster which is capable of running various applications whether in industry or academia. The cloud cluster consists of twenty-one personal computers, including one master node, are linked by a gigabyte switch.

The PCs we use are equipped with AMD Phenom™ II X6 2.8 GHz CPU, 4GB DRAM, 2 TB SATA hard disk, gigabyte NIC, 64-bit Windows7 Enterprise , Apache 2.2.15-x64-openssl-0.9.8m.msi1 and PHP 5.3.8-Win32-VC9-x64 [15]. Each of them has the MapReduce Runtime System we developed in it, as shown in Fig 6. Nodes connect to each other with virtual IP and the master node has an additional public IP for communicating with users.

We compare DIMR with other two different MapReduce frameworks. They are traditional MapReduce runtime, aka MR, which stores data in its

local disk, such as Hadoop and IMR, which stores data in memory, such as Twister. To simplify the experimental environment, we use PHP programing language to develop the MapReduce Runtime System. We put random-sized data in DIMR system with four different types of algorithm. They are K-Means, PSO(Particle swarm optimization), SA (Simulated Annealing) and GA(Genetic algorithm).

Component	Specification
CPU	AMD Phenom(tm) II X6 1055T Processor 2.80 GHz
RAM	4 GB
HD	2 TB
NIC	NVIDIA Nforce 10/100/1000 Mbps Ethernet
OS	Windows 7 Enterprise 64 Bit
Software	Apache 2.2.15-x64-openssl-0.9.8m.msi
	PHP 5.3.8-Win32-VC9-x64

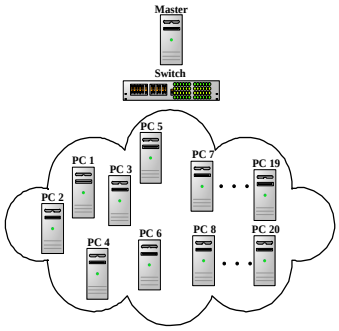

Figure 6 Hardware specifications and network topology of experimental environment

5.2 Previous Optimization Algorithms

K-Means clustering is a method of cluster analysis which aims to partition n observations into k clusters. Each Map function of K-Means will execute complex computation, and produce unclassified cluster intermediate data of key/value pairs. Each Reduce function of K-Means collects intermediate data simply according to the integer of group classification, and finally produces the output.

PSO is an optimization method based on the population dynamics simulation, its concept comes from social behavior. The individual behavior will not only be influenced by the past experience and cognation, but also by the behavior of whole society. According to the past experience, and each PSO has their speed in each node, they will adjust search strategy. It is shown that PSO can quickly identify the optimal solution from many search results, and provides a high degree of adaptability to optimize the dynamic system. The Map function in PSO application will be initialized for each node, after that Reduce function will find the best answer through collecting information from each node.

SA is an approximate solution which is commonly used to solve the optimization problem, and it is according to the principles of statistical thermodynamics. The SA solution is a phenomenon that during the annealing process, the analog material will reach the low-temperature state itself, and it has developed into an optimization solution. Due to the simple search, and it has the ability to jump off the local minima, so it has successfully solved many optimization problems. Map function in SA executes the action of mutation, and it will generate new value. Next, transfer the value generated by last Map function to Reduce function. Reduce function in SA select the optimal solution, and provide the parameters to next Map function.

GA is an algorithm that imitates the evolution of sexual reproduction mechanism to use mechanisms such as mating and mutation. The GA's performance is quite excellent that has been widely used in various areas of AI, especially in the optimization problem. In every generation, any population will become the best one because of the great adaption through natural selection or mutation of the new life. GA is one of the evolutionary algorithms and used to solve optimization search algorithm in computational mathematics generally. The Map function in GA executes cross over and mutation, and transfer the results to the Reduce function. The Reduce function in GA selects the optimal solution, and provides the parameters to next Map function.

5.3 Impact of Map and Reduce Number

In our experiments, the size of input data is 1GB and the ratios of Map and Reduce, are 5 Maps to 15 Reduces, 10 Maps to 10 Reduces and 15 Maps to 5 Reduces, respectively. These ratios affect the execution time of the selected applications under three different types of MapReduce framework. The system performance for the algorithms K-Means, PSO, SA, and GA under different MapReduce framework are evaluated in our experiments.

In Fig. 7, we can observe the performance of K-Means used DIMR, MR, and IMR, respectively. We found that the system will gain higher performance when the number of Map functions is increased. The reason is that the Map function executes more complex works than the Reduce function. Therefore, even we decrease the number of Reduce functions; the system performance can still be improved by increasing the number of Map functions.

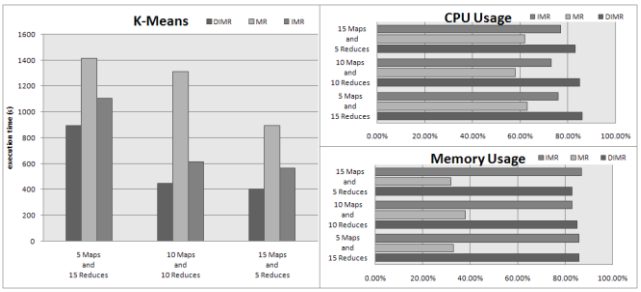

Figure 7 System behaviors for K-Means

K-Means in DIMR has the best performance compared to other MapReduce runtime, because the data stored in memory, not stored in disk, i.e., the time required for I/O is decreased. Our dynamic scheduler in DIMR decides the resource distribution and utilization to make every node work efficiently.

As shown in Fig. 8(CPU usage), the CPU average usage in DIMR is higher than other types of MapReduce runtime. We can also know that the whole computing time is shorter, because DIMR uses a lot of memory space at each node in DIMR. Memory stores the input file and the output file completed to reduce the time of accessing data from remote node.

K-Means in MR has poor performance because it uses the remote file system to store the input file, output file, and the result of calculation. We can know that the MR does not utilize the computation resource at each node efficiently as shown in the CPU Usage and Memory Usage of MR in Fig. 7. Since the data is stored in file system in MR, it will spend more time to transfer the data to memory for computation and makes the nodes idle for a long while. This leads to inferior system performance.

The performance of K-Means in IMR is below the DIMR but better than the MR, since IMR also uses memory to store the input file and output file. However, the system architecture does not distribute the task by considering the state of each node, and it still need to transfer data through the network, rather than in the same machine. Therefore, the algorithm will increase the network I/O, and to degrade the performance of overall system.

Fig. 8 shows the execution time, CPU usage, and memory usage of PSO in DIMR, MR, and IMR respectively. In Fig. 8, we can observe that increase the number of Map functions can reduce the time of system execution efficiently. The reason is that the result of PSO computation is transferred to the memory directly at each node. The DIMR has better performance when the number of Map functions is over fifteen, because the DIMR can dynamically change the working state, it means that the role of a Work Node in DIMR can switched to be a Map Node or a Reduce Node to process different function. Due to PSO has specific acceleration parameters, the more number of Map functions increased, the faster answer of PSO converges. DIMR saves the computation time and network resources by storing data in memory and dynamic node assignment to achieve the best system performance.

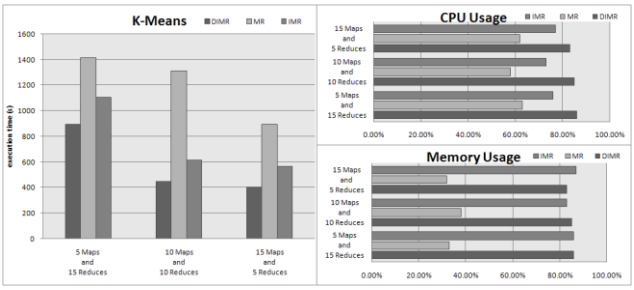

Figure 8 System behaviors for PSO

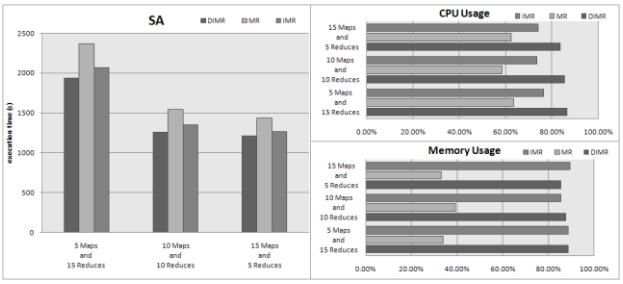

Figure 9 System behaviors for SA

Fig. 9 shows the SA application performance of 1 GB input file under different processing methods, DIMR, MR, and IMR with different number of Map functions and Reduce functions. We found that the process time of SA application can be effectively reduced when the number of Map functions is increased, and CPU and memory space are highly utilized. We also noticed that the system performance is almost the same when the number of Map functions is ten and fifteen, because the Reduce function needs to refresh and find the best temperature parameters. The Reduce function of SA performs much work and needs more computation resource than that of other algorithms. Therefore, the performance is improved a little bit when the number of Map functions is fifteen and the number of Reduce functions is five. Compared to K-Means and PSO applications, SA application requires a longer operation time and more judgment conditions to converge the best answer correctly; therefore, the performance of SA application is slightly lower than that of K-Means application and the PSO application.

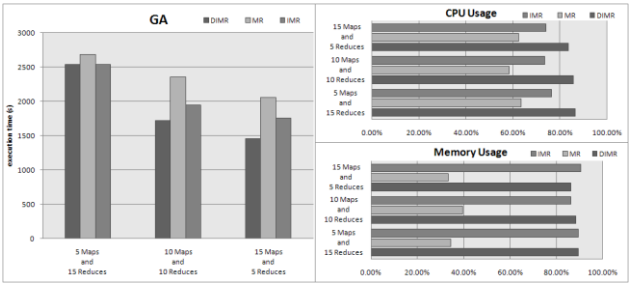

Figure 10 System behaviors for GA

Fig. 10 shows the GA application performance with 1 GB input file under using different processing runtime, DIMR, MR, and IMR, in different number of Map functions and Reduce functions. We find that the GA program has better performance when the number of Map functions is increasing, because the Map function handles complex calculations and the Reduce function handles simple analysis results. Compare to other algorithms, the GA takes more processing time under using different processing methods, DIMR, MR, and IMR, because the GA application needs to broadcast a lot of information, and therefore consume more processing time and network resources. The GA application transfers a large number of messages to all Nodes for cross-over, mutation, or other judgment methods in each iteration and decides the generation parameters for survivals. However, GA application can achieve better performance using DIMR processing method due to the dynamic task allocation and avoid storing data in the disk method, and significantly reduce the I/O bottleneck.

5.4 Resource Behavior for DIMR, MR and IMR

We compare the network I/O and disk I/O performance among the proposed method DIMR, IMR, which stores dada in memory rather than stores data in disk and sends data over network, and traditional MR. In this subsection, we describe the disk operation times and network resource under different operator methods, DIMR, MR, and IMR, in different algorithms, K-Means, PSO, SA, and GA.

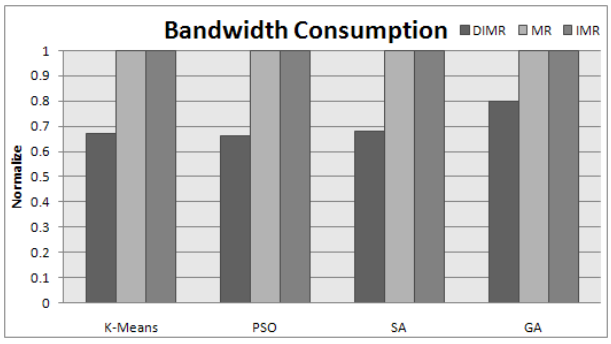

Figure 11 Bandwidth Consumption

Fig. 11 shows the bandwidth consumption after normalization under different operator methods, DIMR, MR, and IMR, in different algorithms, K-Means, PSO, SA, and GA. It can be seem from Fig. 11 that MR and IMR consume the same bandwidth in different algorithms, because MR and IMR will send the intermediate data and output data to the next node to compute. The MR and IMR have no ability for the node to switch various works, and fix the task category for each node, called Map function or Reduce function. Different from two other approaches, DIMR will decide the task category automatically, and turn the node, which keeps the most key value, to the next Phase task category in local area. Therefore, DIMR can reduce the bandwidth consumption and avoid the network I/O Bottleneck and increase performance.

6 Conclusions

With Map function and Reduce function, MapReduce can provide a high performance and simple operating environment. The traditional MapReduce spend too much effort on waiting Map and Reduce to store the information into file system, which makes it unsuitable for applications with high ratio data transfer. In this paper, we propose a novel MapReduce Runtime System – DIMR. Instead of using disk, storing data in memory not only reduce the execution time, but also solve the bottleneck problem caused by disk I/O. DIMR lowers the network I/O and disk I/O ratio compared to the traditional MapReduce runtime and greatly improve the system performance.

Acknowledgements

This work was supported partially by the National Science Council of Republic of China under grant NSC 99-2221-E-032-032-MY3 and NSC 102-2221-E-032-027.

References

[1] A. Fox and R. Griffith, “Above the clouds: A Berkeley view of cloud computing”, Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Tech. Rep. UCB/EECS, vol. 28, Feb 2009.

[2] K. Li, L. T. Yang and X. Lin, “Advanced topics in cloud computing”, *Journal of Network Computer Application*s, Vol. 34, Issue 4, July 2011, pp. 1033-1034.

[3] A. Liu, D.M. Batista and M. Alomari, “A Survey of Large Scale Data Management Approaches in Cloud Environments”, *Journal of the IEEE Communications Surveys & Tutorials*, Vol. 13, Issue 3, 2011, pp. 311-336.

[4] J. Dean and S. Ghemawat, “MapReduce : Simplified data processing on large clusters.”, *In Proceedings of the 6th Symposium on Operating Systems Design and Implementation* (OSDI), December 2004, pp. 137–150.

[5] W. Dai , C. Jiao and T. He, “Research of K-means Clustering Method Based on Parallel Genetic Algorithm”, *In Proceedings of Intelligent Information Hiding and Multimedia Signal Processing*, Nov. 2007, pp. 158-161.

[6] A.W. McNabb, C.K. Monson, and K.D. Seppi, “Parallel PSO using MapReduce”, *In IEEE Congress on Evolutionary Computation*, Sept. 2007, pp. 7-14.
[7] G.S. Sadasivam, and D. Selvaraj, “A novel parallel hybrid PSO-GA using MapReduce to schedule jobs in Hadoop data grids”, *In Proceedings of Nature and Biologically Inspired Computing (NaBIC)*, Dec. 2010, pp. 377-382.
[8] J. Song, and W. Yi, “Improvement of original particle swarm optimization algorithm based on simulated annealing algorithm”, *In Proceedings of International Conference on Natural Computation (ICNC)*, May 2012, pp.777-781.
[9] T. White, “*Hadoop: The Definitive Guide*”, ISBN: 978-0-596-52497-4, O'Reilly Media, Yahoo! Press, June 5, 2009.
[10] D. Borthakur, “The hadoop distributed file system: Architecture and design”, Hadoop Project Website, 2007.
[11] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, “Twister: a runtime for iterative mapreduce”, *In Proceedings of HPDC*, 2010, pp. 1-9.
[12] C. Jin, C. Vecchiola and R. Buyya, “MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms”, *In Proceedings of IEEE International Conference on eScience*, Dec. 2008, pp. 214-221.
[13] G. S. Sadasivam and D. Selvaraj, “A Novel Parallel Hybrid PSO-GA using MapReduce to Schedule Jobs in Hadoop Data Grids”, *In Proceedings of World Congress on Nature and Biologically Inspired Computin*g, Dec. 2010, pp. 377-382.
[14] S. Ibrahim, H. Jin, L. Lu, B. He and S. Wu, “Adaptive Disk I/O Scheduling for MapReduce in Virtualized Environment”, *In Proceedings of International Conference on Parallel Processing* (ICPP), Sept. 2011, pp. 335-344.
[15] X. Yu and C. Yi, “Design and Implementation of the Website Based on PHP & MYSQL”, *In proceeding of International Conference on E-Product E-Service and E-Entertainment (ICEEE)*, Nov. 2010, pp. 1-4.[16] Ssu-An Lo, Chiun-Chieh Hsu, Shu-Ming Hsieh, Wei-Ming Chen, “RankCloud: A Cloud-Based Webometrics Ranking System”, *Journal of Internet Technology*, Vol. 14 No. 1, P.
[17] Seokil Song, Ki-jeong Khil, Yun Sik Kwak, Daesik Ko, Seung-Kook Cheong, “ Software RAID for Data Intensive Applications in Cloud Computing”, *Journal of Internet Technology*,Vol. 14 No. 3, P.529-534.
[18] Tin-Yu Wu, Chi-Yuan Chen, Ling-Shang Kuo, Wei-Tsong Lee, and Han-Chieh Chao, "Cloud-based Image Processing System with Priority-based Data Distribution Mechanism", *Computer Communications*, Vol. 35, No. 15, pp. 1809-1818, September 2012.
[19] Shen Wang, Xiamu Niu, “Cover the Trace of Image Forgery by PSO”, *Journal of Internet Technology*, Vol. 14 No. 1, P.161-168
[20] Yanfeng Zhang, Xiaofei Xu, Yingqun Liu, Xutao Li Yunming Ye1, “A Novel Decision Cluster Classifier with Nested Agglomerative K-Means”, *Journal of Internet Technology*, Vol. 14 No. 1, P.145-152

Wei-Tsong Lee received his B.S., M.S. and Ph.D degrees in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan. In 2003, he joined the department members of Electrical Engineering of Tamkang University and he is currently the chairman of the Department. His research interests are computer architecture, micro-processor interface and computer networks.

Ming-Zhi Wu received the B.S and M.S degree in electrical engineering from Tamkang University. His research interests include cloud computing, .

Hsin-Wen Wei received the Ph.D. degree in computer science from National Tsing Hua University. She is an assistant professor in the Department of Electrical Engineering at Tamkang University. Her research interests include cloud computing, wireless networks, real-time systems, and graph theory.

Fang-Yi Yu received the B.S and M.S degree in electrical engineering from Tamkang University. His research interests include image processing, multimedia streaming, cloud computing.

Yu-Sun Lin received the B.S and M.S degree in electrical engineering from National Chung Hsing University. His research interests include image processing, multimedia streaming, cloud computing.