Contents lists available at SciVerse ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Cloud-based image processing system with priority-based data distribution mechanism

Tin-Yu Wu [a], Chi-Yuan Chen [b], Ling-Shang Kuo [c], Wei-Tsong Lee [c], Han-Chieh Chao [a,b,d,*]

[a] *Institute of Computer Science & Information Engineering, National Ilan University, Taiwan, ROC*
[b] *Department of Electrical Engineering, National Dong Hwa University, Taiwan, ROC*
[c] *Department of Electrical Engineering, Tamkang University, Taiwan, ROC*
[d] *Department of Electronic Engineering, National Ilan University, Taiwan, ROC*

ARTICLE INFO

Article history:
Available online 20 July 2012

Keywords:
3D image
Cloud system
Multicast streaming
Image processing

ABSTRACT

Most users process short tasks using MapReduce. In other words, most tasks handled by the Map and Reduce functions require low response time. Currently, quite few users use MapReduce for 2D to 3D image processing, which is highly complicated and requires long execution time. However, in our opinion, MapReduce is exactly suitable for processing applications of high complexity and high computation. This paper implements MapReduce on an integrated 2D to 3D multi-user system, in which Map is responsible for image processing procedures of high complexity and high computation, and Reduce is responsible for integrating the intermediate data processed by Map for the final output. Different from short tasks, when several users compete simultaneously to acquire data from MapReduce for 2D to 3D applications, data that waits to be processed by Map will be delayed by the current user and Reduce has to wait until the completion of all Map tasks to generate the final result. Therefore, a novel scheduling scheme, Dynamic Switch of Reduce Function (DSRF) Algorithm, is proposed in this paper for MapReduce to switch dynamically to the next task according to the achieved percentage of tasks and reduce the idle time of Reduce. By using Hadoop to implement our MapReduce platform, we compare the performance of traditional Hadoop with our proposed scheme. The experimental results reveal that our proposed scheduling scheme efficiently enhances MapReduce performance in running 2D to 3D applications.

1. Introduction

Cloud computing is composed of servers that provide high storage capacity, high flexibility and high-computing performance [1–5]. Because a cloud computing system has several computing servers, users are able to execute procedures that require large amounts of computing resources, process a great deal of data on the cloud, and complete the tasks by low-cost and low-power devices. To process the procedures requested by users with cloud computing, users can simultaneously enhance the performance of the procedures. By using cloud computing for knowledge discovery and data integration [6,7], Google thus proposed the MapReduce programming model.

As a distributed parallel computing architecture presented by Google, MapReduce automatically executes parallel computing and partitions the data input by the developer. To simply write Map and Reduce functions, the developer can execute a full set of computing procedures. MapReduce mainly include three functions: Master, Map function and Reduce function. The Master periodically obtains the status of the servers that run Map and Reduce functions. The Master next divides the input data equally based on the number of the Map function and distributes the sub-tasks to the designated servers. The Map function converts the data designated by the Master to the intermediate data while the Reduce function merges the intermediate data processed by the Map function together and creates the final result.

Derived from an implementation of Google's MapReduce [8], Hadoop consists of two chief components: Hadoop MapReduce and Hadoop Distributed File System (HDFS). Hadoop MapReduce is a distributed computing platform that includes JobNode, which is responsible for allocating the developer's tasks to the TaskNode and is thus called the Master, and TaskNode, which is responsible for executing the tasks designated by the JobNode, including the Map and Reduce functions written by the developer. HDFS mainly comprises NameNode and DataNode. The DataNodes are responsible for storing the results completed by the Map and Reduce functions while the NameNode is responsible for establishing the index stored in the DataNodes and recording the positions of the data needed for the Map and Reduce functions to execute tasks.

* Corresponding author at: Institute of Computer Science & Information Engineering, National Ilan University, Taiwan, ROC. Tel.: +886 39357400x251.
E-mail address: hcc@niu.edu.tw (H.-C. Chao).

0140-3664/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comcom.2012.06.015

At present, most applications built on Hadoop are low-complexity and high-density computing, like WordCount, Sort and so on [9–11]. While running such applications, the Map function can get the processed result within a very small amount of time and the Reduce function mainly spends time waiting for the data processed by the Map function and integrating the data. However, using MapReduce to implement high-complexity and high-density 2D to 3D image processing process [12,13] may cost lots of time for the Map function to analyze images and to convert the images to intermediate data for the Reduce function to generate 3D images. This paper implements MapReduce on an integrated 2D to 3D multi-user system, in which the Map function is responsible for processing 2D effects, like gray scale [14], sobel [15], Gaussian blur and corner detection [16], and converting the images to intermediate data, while the Reduce function is responsible for integrating the intermediate data generated by the Map function, using 3D model construction algorithm [17] to construct the model of the object, and presenting 3D images according to user requirements.

Traditionally, the Map function converts the images to intermediate data so that the Reduce function can combine the images together for the final output. However, if the computing speed of the Map function is too slow, the data will not be completed and the Reduce function has to wait until the data is completed for further combination. In such a manner, when the computing speed of the Map function cannot cope with the Reduce function, the tasks of the Reduce function will be greatly delayed, which moreover increases the operation time.

In this paper, because large amounts of users simultaneously uses MapReduce for 2D to 3D image processing, we present a solution to the above-mentioned problem to enhance the performance of MapReduce in executing 2D to 3D image processing: Dynamic Switch of Reduce Function (DSRF) algorithm, a scheduling scheme on the Reduce function for users who compete simultaneously to acquire Reduce resources to finish the tasks efficiently. With a priority mechanism, DSRF algorithm allows the Reduce function to switch to different tasks according to the achieved percentage of the tasks. In other words, when a combination task is not finished and the Reduce function is waiting for the Map function to generate intermediate data, the Reduce function can switch to another task to combine the image data first. For example, when the data needed for the first task is not completed yet but the image data of the second combination task is ready, the Reduce function processes the second task first. After finishing the second combination task, supposing the image data of the first and the third tasks is both ready, the Reduce function processes the first task according to the priority values. With our proposed scheme, the time spent waiting for data can efficiently utilized by the Reduce function and the delayed tasks also can be decreased to enhance the performance of MapReduce in 2D to 3D image processing.

The rest of the paper is organized as follows. Section 2 introduces the background and Section 3 presents our proposed system architecture and DSRF algorithm. Section 4 includes the implementation and performance analysis awhile Section concludes this paper.

2. Background

2.1. MapReduce

Proposed by Google, MapReduce is a distributed parallel computing architecture designed for processing large amounts of intensive data and its storage system is Google File System (GFS). For users to process large amounts of data while searching for data by the Google search engine, Google MapReduce provides a simple programming interface for the developer to develop search engine and processing searching procedures. MapReduce is composed of three major components: Master, Map function and Reduce function. The Master is responsible for managing the back-end Map and Reduce functions and offering data and procedures to them. The Map function converts the data to intermediate data so that the Reduce function can combine the intermediate data together to obtain the final output.

Whenever the data is received and processed by the Map and Reduce functions, there will be a record that includes a key and a value. The key generated by the input processed by the Map function refers to the correlation between the data while the value on the other hand means the processed message. When a task is divided into several small tasks and handled by the Map function, their correlations are not marked especially. But, one or several records are generated, including the key and the value, after the Map function finishes processing the designated data. According to the key produced by the Map function, the Reduce function collects the data with the same key together and generates a value to form a new set of key/value.

As shown in Fig. 1, the input file is cut into M chunks before the MapReduce operation and the file size of each chunk ranges between 16 MB to 64 MB. The file chunks are copied to the file system and one of the chunks is especially designated to the Master to find the suitable idle server as the computing server. Next, after the Master determines the servers responsible for the Map and Reduce functions, M Map functions and R Reduce functions are designated to the servers. According to the key/value of the input data, servers responsible for the Map function deliver the processed data to the developer's Map function and store the Map result in the file system. The intermediate data output by the Map function is cut into R pieces and their locations are returned to the Master. Once receiving the locations of the data processed by the Map function from the Master, servers responsible for the Reduce function read the data remotely, sort the keys and group the values with the same key. The Reduce function calculates the number of the single key and forwards the data to the developer's Reduce function to generate the final output. After the data is completely processed, the Master notifies users to access the result.

Because of the high-speed network and a storage system that can store a large volume of data, MapReduce allows most search engines to operate. The hardware of MapReduce belongs to an open-source operating system and is controlled and maintained by managers to reduce not only the cost of purchasing and maintaining equipments but also the cost of developing MapReduce. Different from centralized RAID-based SAN and NAS storage

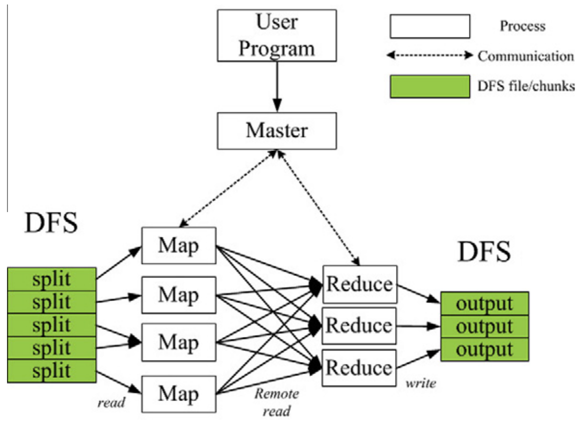

Fig. 1. MapReduce framework.

systems, the GFS of MapReduce belongs to RAIN (Redundant Arrays of Independent Nodes) architecture [18]. Every MapReduce node has its local storage space, each of which is combined with one another by the network to form a virtual storage system. Thus, the storage space of the file system can be easily expanded by adding the nodes. During the process of adding nodes, MapReduce continues to work and the task will not be terminated. Owing to its high scalability, the Map and Reduce functions can be executed on thousands of and even more nodes. However, the nodes' hardware is not exactly the same and thus some nodes cannot present stable performance and might be out of order at any time. By using simple mechanisms to replicate data and perform the backup, GFS and MapReduce can maintain the operation. To repair the damaged nodes, the manager removes the damaged ones and repair the hardware. New nodes can join in the computing cluster any time and the Master takes charge of controlling the operation and performing the backup.

2.2. Hadoop

Because Google proposed the concept of MapReduce without presenting the MapRedcue platform, Apache Software Foundation therefore developed Hadoop, a software framework to implement the MapReduce programming model. Written in the Java language, Hadoop allows users to process and compute large amounts of data. In addition, similar to Google's GFS, Hadoop provides a filesystem called HDFS (Hadoop Distributed File System).

2.2.1. Hadoop MapReduce

In Hadoop MapReduce, the Map and Reduce functions are interlinked and coexist. In Fig. 2, when a program processes the data, like Google's MapReduce, the Master divides the data and distributes the small tasks to the servers responsible for the Map function written by the developer. After the data is processed completely, the Map function stores the data into one or more key/value pairs for the Reduce function to identify and process, namely the intermediate data. Next, the Master transmits the storage locations of the intermediate data to the Reduce function to find out the data to be integrated, create one or more new key/value pairs, and report back to the Master the completion of the task. Finally, the Master notifies users to access the result from the file system. The Master spontaneously designates different nodes to take charge of the Map and Reduce functions: JobNode and TaskNode, respectively.

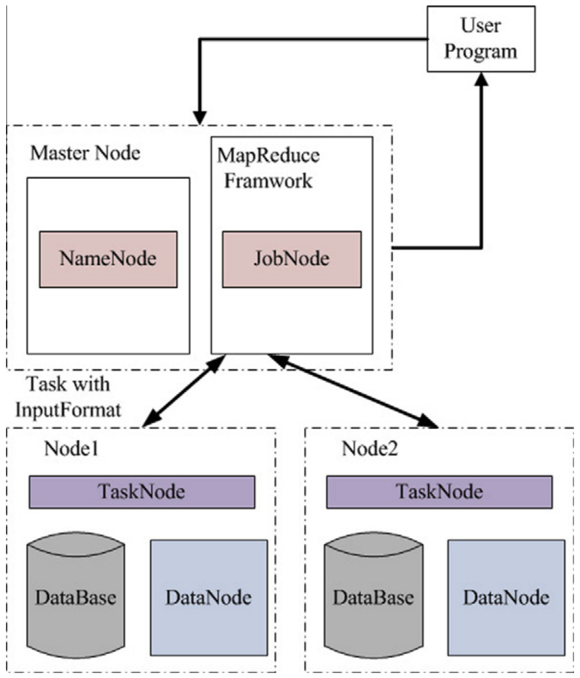

Fig. 2. Hadoop MapReduce framework.

As the manager in Hadoop MapReduce, the JobNode is responsible for allocating the tasks to the computing servers. As mentioned previously, the Master designates the tasks according to server status and the task designation is exactly the responsibility of the JobNode. When the JobNode is informed of the MapReduce function to be execute, the JobNode allocates the tasks based on the operation status reported regularly by the servers.

As the executor in Hadoop MapReduce, the TaskNode is responsible for executing the tasks designated by the JobNode. During the task execution process, the TaskNode continues to report its operation status. Supposing the TaskNode is damaged, the JobNode receives the message and find another idle server to run the task originally executed by the TaskNode.

2.2.2. HDFS

HDFS, Hadoop Distributed File System, is designed to integrate distributed storage resources into a file storage system with high fault-tolerance, high performance and mass storage volume. In Hadoop, massive amounts of data and files generated during the computation are stored in this file system. Although HDFS can only process files using HDFS shell, Hadoop series systems, like MapReduce, can integrate with HDFS and use HDFS for data storage and backup. Like MapReduce, HDFS chiefly comprises NameNode and DataNode, each of which is defined below.

The NameNode, the file manager in HFDS, is responsible for managing and storing the attributes and permission of the files in the system. Because the NameNode does not store files, it notifies MapReduce of the storage locations and permission of the files when MapReduce demands the files.

In HDFS, the DataNodes are responsible for storing files. After a file is split into many blocks, each block is copied to a DataNode and the duplicate of the blocks are also stored by other DataNodes to ensure the completeness of the data in the system when any of the DataNodes is broken. Moreover, the DataNodes inform the NameNode of the files' storage locations so that the NameNode can offer correct information to MapReduce.

2.3. Image processing procedures

Fig. 3 displays the flowchart of model reconstruction. To process two images, we first use Harris corner detector [16] to calculate the interest points, complete the matching by sum of absolute difference (SAD), and label the matched interest points. Next, we find out the thresholds of data, which is processed by Sobel marginalization and Gaussian Blur. Because of the intense edge information of the data after edge binarization, the relationship among the interest points can be figured out.

2.3.1. Harris corner detector

Common corner detection methods include Harris, Susan, Zheng and Harr. Among these methods, Harris corner detector is utilized most often because of its higher accuracy compared with other existing corner detection methods. Based on the points of interest proposed by Moravec in 1977 [2], Harris corner detector was proposed with the attempt to modify the problem of Moravec algorithm that the corner might be located at the edge. Before introducing Harris algorithm, we first introduce Moravec corner detection algorithm briefly.

A square mask with point P as the center is established. Supposing V, the gray variation of the mask, is higher than the defined

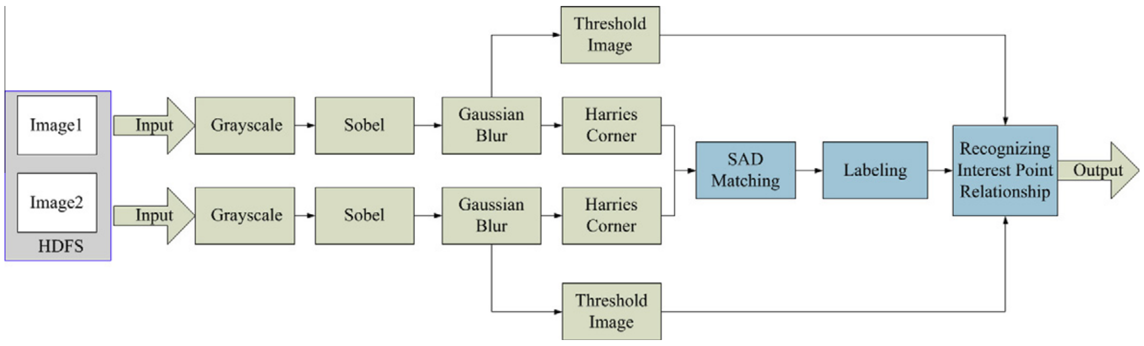

Fig. 3. Flowchart of model reconstruction.

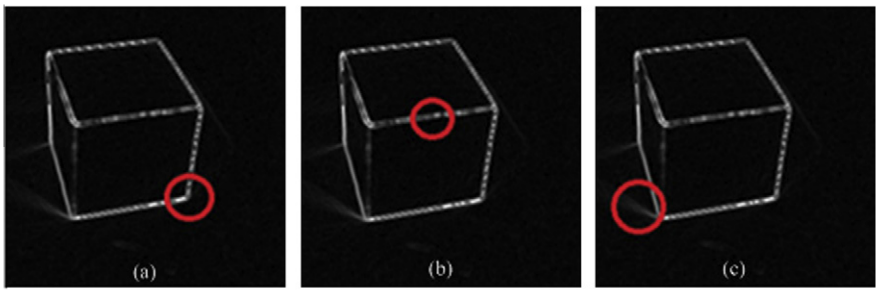

Fig. 4. Moravec corner detection.

threshold, point P is defined as the corner (Fig. 4(a)). Moravec algorithm can be expressed by mathematical function as

$$E_{(x,y)} = \sum_{u,v} w_{(u,v)} |I_{(x+u,y+v)} - I_{(u,v)}|^2, \tag{1}$$

where w refers to the image window, which means to retrieve only the data inside of the mask during the search. Data outside of the mask does not influence the computing result. $I_{(u,v)}$ symbolizes the gray level of location (u, v). However, this method often leads to misjudgments because of higher gray variation occurred at the edge of the object (Fig. 4(b)), or caused by noises (Fig. 4(c)). Moreover, it is very difficult to establish the threshold in a complicated environment.

To solve the problems of Moravec: the difficulty in establishing the thresholds, the misjudgment in determining the object's edges, and the inability to recognize the corners after rotating the object, Chris Harris and Mike Stephens proposed Harris corner detection algorithm in 1988.

2.3.2. Sum of absolute difference (SAD)

To subtract pixels of two image blocks, SAD can measure the similarity between image blocks. The lower SAD value means the higher similarity of the two blocks. SAD can be expressed by the mathematical function as

$$SAD = \sum_{x_A \in \Omega_{A,B}} |A(x_A) - B(x_A)|, \tag{2}$$

where A and B stand for the two images to be judged, and $\Omega_{(A,B)}$ means that there are N pixels of A and B in the overlap area. Thus, we use SAD to judge the relationship of interest points detected by Harris corner detection method, and label the corresponding points for computers to judge the corresponding locations of interest points after rotating an object. With the relationship of the corners, we can use interpolation to judge the location of the corner while being photographed to simulate the object rotation.

2.3.3. Model reconstruction

With the simulated corner location only, it is impossible to judge the shape of an object in human vision. Therefore, the next step is to reconstruct the edge relations between corners.

By binarizing the image processed by Sobel marginalization and Gaussian Blur, we can get the intense edge information with low noises to determine whether the two points have edge relations. To define the average brightness of two points as B, the distance as L, a random point between two points as P, we can get the mathematical function as

$$B = \sum_{n=1}^{L} P_{(n*dx,n*dy)}, \tag{3}$$

where

$$dx = -\frac{x_1 - x_2}{L}, \tag{4}$$

and

$$dy = -\frac{y_1 - y_2}{L}. \tag{5}$$

When B is higher than the defined threshold, it can be determined that the two points have edge relations.

3. DSRF (Dynamic Switch of Reduce Function) algorithm

3.1. MapReduce runtime system with 2D to 3D application

First, a Master is established in the server cluster to acquire the operation status of all servers and the Map and Reduce functions written by the developer for 2D to 3D application are offered to the Master. To connect all servers via the network, the Master accesses the resources of all servers' HDFS and the current operation status of Map and Reduce, and allocates the Map and Reduce functions to designated servers. In other words, the Master masters the resources and status (idle or working) of all servers. Because the Master's NameNode owns the storage locations of the data stored in HDFS, the Master is responsible for notifying the computing servers of the locations of the data needed to execute Map and Reduce functions. By communicating with the Master, the developer can know the operation status of all servers and the data in the servers, choose the maximum limit of designated servers, and easily allocate the Map and Reduce function to the servers.

2D to 3D application is divided and handled separately by the Map function and Reduce function. In Fig. 5, the Map function gives service to object cameras and provides the intermediate data for the Reduce function. The Map process includes grayscale, sobel, Gaussian blur and corner detection. Through the four image processing procedures, we can get the positions of the corners and the brightness of the original images from two cameras. In this paper, four image processing algorithms are moreover written into the Map function for the Master to allocate. Next, the Reduce function accesses the Map function to get the image data to be processed and creates the final output by SAD matching and 3D graphing. The Reduce function includes SAD matching, labeling and model reconstruction. The Reduce function combines the data of two images, like the positions of the corner and motion vector, etc., and simulates the graphs of all angular vector by model reconstruction. Finally, through wireless network, the output is sent back to the terminal devices of MapReduce for parameter adjustments, and users are able to access the needed angles of vision by HDFS and display the images by terminal devices.

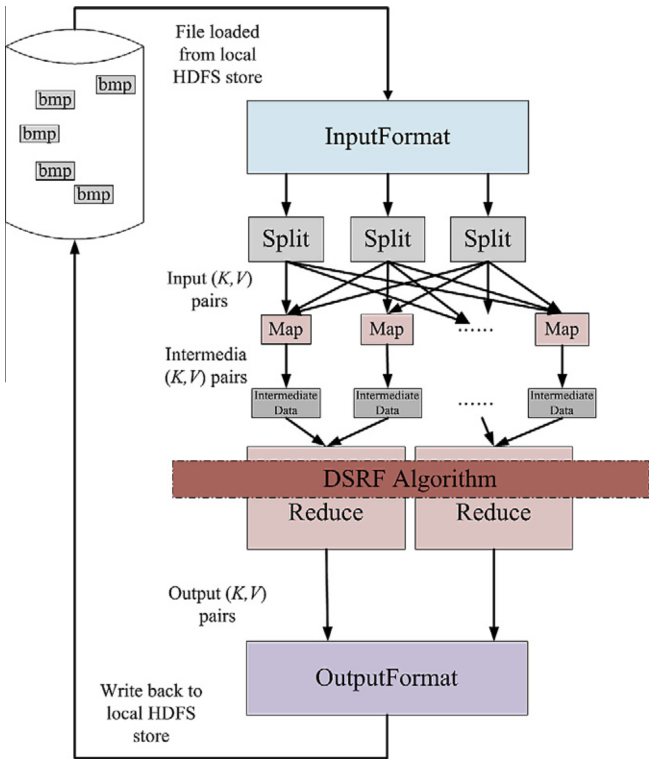

Fig. 6. DSRF algorithm.

3.2. DSRF algorithm

As shown in Fig. 6 and Algorithm 1, we use Hadoop to establish the computing environment, which includes MapReduce and HDFS. The mechanism of MapReduce is that the developer adapts the 2D to 3D application to a framework of Map and Reduce functions and puts the framework in the Master server. Thus, before executing MapReduce, the developer can allocate the functions to the TaskNodes through the JobNode in the Master without spending additional time allocating Map and Reduce functions. The image data obtained by the monitors are stored in key/value pairs and replicated to the DataNodes of Hadoop HDFS. After the DataNodes store the images, the locations of the images will be sent

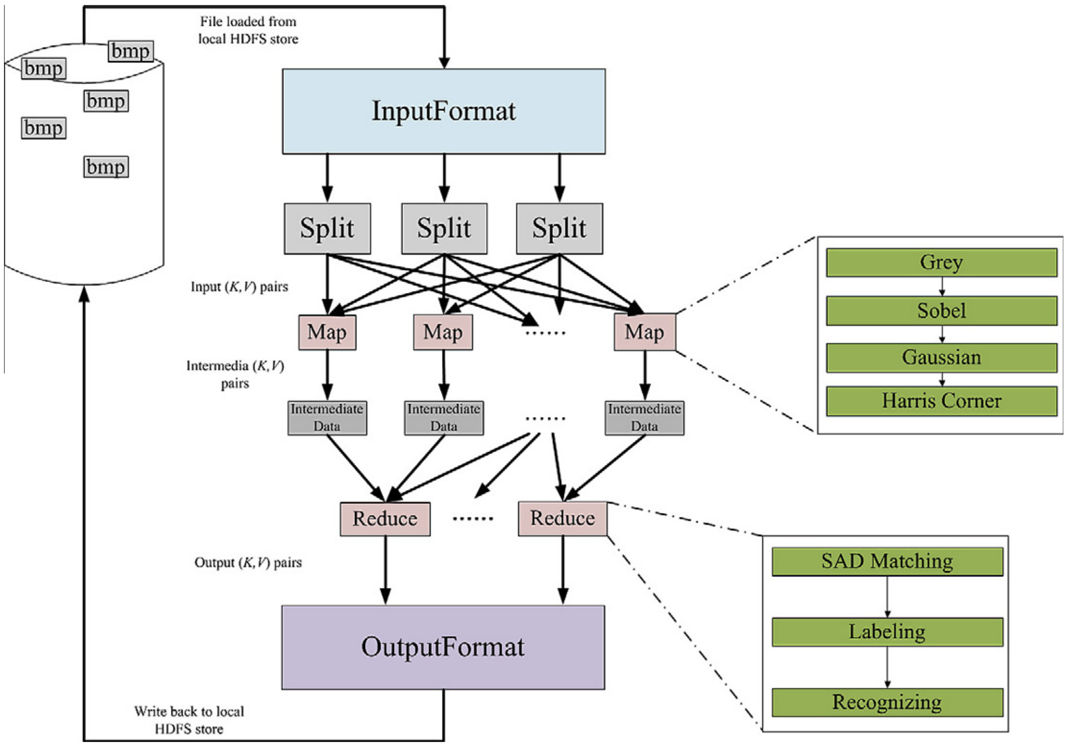

Fig. 5. MapReduce runtime system with 2D to 3D application.

to the NameNode in the Master. Since the back-end servers periodically send their busy status to the Master, the Master receives the commands, searches for idle servers, and designates the Map and Reduce functions to servers by the JobNode. The Master moreover finds out the locations of the data needed to execute functions by the NameNode and informs the servers responsible for Map and Reduce functions to access. After the Map function receives the image data and finishes processing, the output framework will be the intermediate data in key/value pairs for the Reduce function to recognize. The intermediate data output by the Map function is stored in HDFS and its location is recorded by the Master. Thus, when the Reduce function gets ready to execute a combination task, the Master informs the servers responsible for the Reduce function of the location of the intermediate data. Once the data is completed, the Reduce starts the combination task and stores the result in key/value pairs in HDFS when the task is accomplished.

Algorithm 1: DSRF

```
1. Initialize image process;
2. Write Map-Reduce structure;
3. Set server to handle Map or Reduce;
4. Write image into HDFS of Hadoop;
5. Repeat image process until image data processed;
6. Set Map-Reduce and data into Map layer cutting by
   managers;
7. When Map complete it's task;
8.    Sent Map output data to Reduce;
9.    If Reduce task cannot get all data from Maps
10.        Set priority value to the task and put it into Reduce
   wait queue;
11.        Detect the priority of task which have all data are in
   the Reduce process queue;
12.        Operate the higher priority task which have all data;
13.    Otherwise
14.        Reduce sent the Reduce output data to clients.
```

Because the Map and Reduce functions spend longer time processing images, the Reduce function may need extra time waiting for the data to be written into HDFS. Supposing the Reduce function is ready to begin a combination task but cannot attain the full intermediate data of the Map function from HDFS, the Reduce function has to wait until all intermediate data needed has been written into HDFS to begin the task, which wastes much more time in processing a combination task. For this reason, our DSRF (Dynamic Switch of Reduce Function) algorithm proposes to process the rest intermediate data first while the Reduce function is waiting for the intermediate data. Our DSRF algorithm adds a scheduling mechanism to the Reduce function and prioritizes the combination tasks to be executed. When the intermediate data needed for the first combination task is not fully written into HDFS, the Reduce function switches to the second combination task with complete intermediate data. After finishing the second combination task, if the intermediate data of the first task is ready, the Reduce function returns to the first task, writes the task result to HDFS and completes the task.

3.3. DSRF algorithm working principle

The Master notifies the servers responsible for the Map function of the locations of the images needed for the Map function. The Map function next accesses the locations, obtains the images to be processed, and processes four procedures: grayscale, sobel, Gaussian blur and corner detection. After the corner detection is completed, the Map function converts the image data to the intermediate data for the Reduce function to execute and informs the Master that the task has been completed. According to the Master's notification, the Reduce function reads the intermediate data for the combination tasks. DSRF algorithm in the Reduce function designates different priority values to different combination tasks based on their arrival time. The earlier combinations tasks have higher priority values and vice versa. Supposing the data needed for the earlier tasks is not ready yet, DSRF algorithm first switches to other combination tasks with full image data. After the Reduce function finishes the current combination task, DSRF algorithm asks the Reduce function to process the combination task with higher priority value and full image data. Finally, the image data completed is written to HDFS and offered to users.

For example, as displayed in Fig. 7, the Reduce function is ready to process the first combination task but split1-5 of the first task is not stored in HDFS yet, which means that the Reduce function does not have the full data for the first combination task. At the same time, the data for the second combination task is ready. In such a manner, the Reduce function chooses to execute the second combination task while waiting for split1-5. After the second combination task is finished and the data for the third and the first combination tasks is both ready, the Reduce function chooses to process the first task according to priority values. By such a scheduling mechanism, our proposed method allows the Reduce function to utilize the time waiting for data, to first process the combination tasks with full data chunks, and to return to the earlier tasks based on priority values. The following implementation will prove that our proposed DSRF algorithm and scheduling mechanism can enhance the system performance.

4. Implementation

4.1. Implementation environment

This section implements our DSRF algorithm in Hadoop MapReduce and Fig. 8 presents the implementation environment. This paper uses the Ubuntu 10.04 as the OS to run Hadoop and adopts Hadoop 0.20.2. Because Hadoop MapReduce is written in the Java language, we use Eclipse IDE to write the Map and Reduce functions. There are totally four servers in the implementation: two for the Map function and the other two for the Reduce function. The four servers are connected by the cable lines with a rate of one gigabit per second. Next, we aim to analyze the influence of user number and the number of Map and Reduce functions to the total time and compare the results with traditional Hadoop.

Left view and right view in Fig. 9 are the images that the Map function will process and convert to a 3D model by image processing procedures. In other words, with the image data of left view and right view, like corners and angles, we can simulate all angles from left view to right view and display the 3D images on terminal devices. The lower models in Fig. 9 are simulated by MapReduce. Thus, through the touch pads of terminal devices, users can view the simulated 3D images at different angles.

4.2. Performance analysis and evaluation

This section makes a comparison of MapReduce with DSRF algorithm and traditional Hadoop in processing 2D to 3D application. Fig. 10 shows the influence of user number to the total computing time in our proposed DSRF algorithm and traditional Hadoop. The horizontal axis means the increasing user number while the vertical axis means the total time for the MapReduce operation. The figure reveals that before the user number reaches 70, DSRF algorithm obviously accelerates the MapReduce operation efficiently and outperforms traditional Hadoop because our proposed

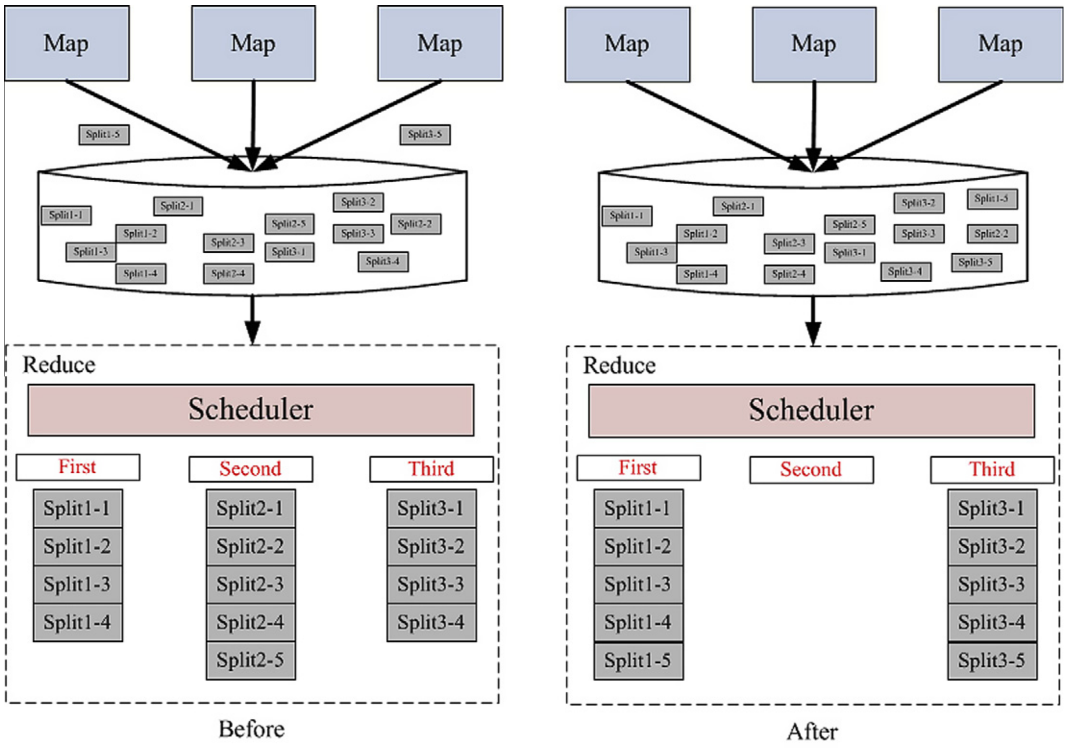

Fig. 7. DSRF algorithm working principle.

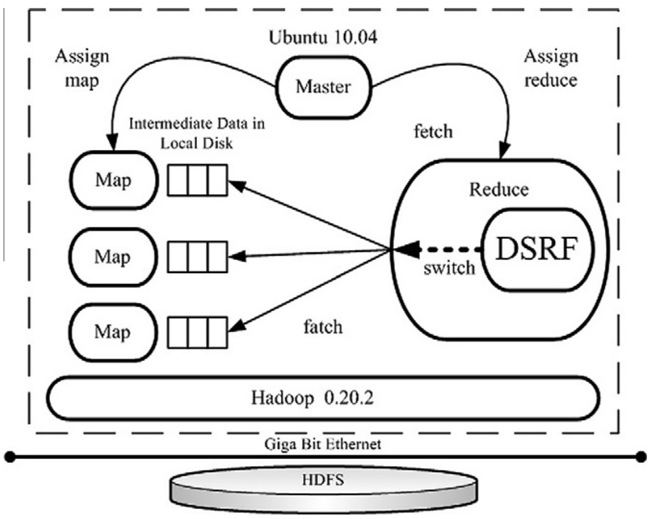

Fig. 8. DSRF in Hadoop.

DSRF algorithm can switch to the combination task that needs to be processed first. The ideal state is that the total time of our DSRF algorithm is times faster than traditional MapReduce. However, when the user number exceeds 70, since it takes time to switch tasks and the delay keeps increasing, our DSRF algorithm spends large amounts of time switching and processing tasks while traditional Hadoop waits for the tasks to be completed, which omits the overhead in switching. Thus, in this implementation environment, when the user number exceeds 70, the user number for our DRSF algorithm to switch soon reaches the saturation point and traditional MapReduce will perform better than our method.

Fig. 11 displays the influence of user number to switch times of DSRF algorithm under the ideal state and in DSRF algorithm. The horizontal axis means the increasing user number while the vertical axis means the switch times of DSRF algorithm. The figure shows that the switch times of DSRF algorithm increases with the user number, also. The ideal state presents the ideal average switch times. When the user number is 10, the average switch times is 2 to 3. When the user number reaches 20, the ideal average switch times is 5. Fig. 11 moreover reveals that to process 2D to 3D application by our proposed MapReduce, the switch times fall near the ideal state when the user number is under 70. Nevertheless, when the user number exceeds 70, the bandwidth is not enough to serve so many users and the switch times gradually deviates from the ideal state.

Fig. 12 shows how the delay time of the combination task develops if the user number increases. When the Reduce function is going to process a combination task but the data is not complete, the Reduce function has to switch to other tasks, which increases not only switch times but also extra time in switching tasks. The figure reveals that the delay time of the system increases with the switch times. Thus, unavoidable switch times will be the problem to be solved in our scheduling mechanism in the future.

Fig. 13 shows how the throughput of the Reduce function increases or reduces with the user number when the system runs for 500s. When the number of tasks handled by the Reduce function increases, the upcoming tasks will be delayed because the data of the former tasks is not fully collected yet. On the other hand, when the tasks must be processed by the Reduce function, too long switching time of DSRF may postpone the execution time. In such a manner, traditional Hadoop will first finish the previous task and then the next combination task. Therefore, the delay of traditional Hadoop exceeds the delay of our proposed DSRF algorithm under certain situations. Fig. 13 reveals that when the user number is 10, 20 or 30, the performance of Hadoop Native and DSRF algorithm is approximately the same. When the user number keeps increasing, the throughput within 500 s gradually reduces with the increase of switch times. However, when the user number exceeds 70, the delay due to the scheduler worsens the perfor-

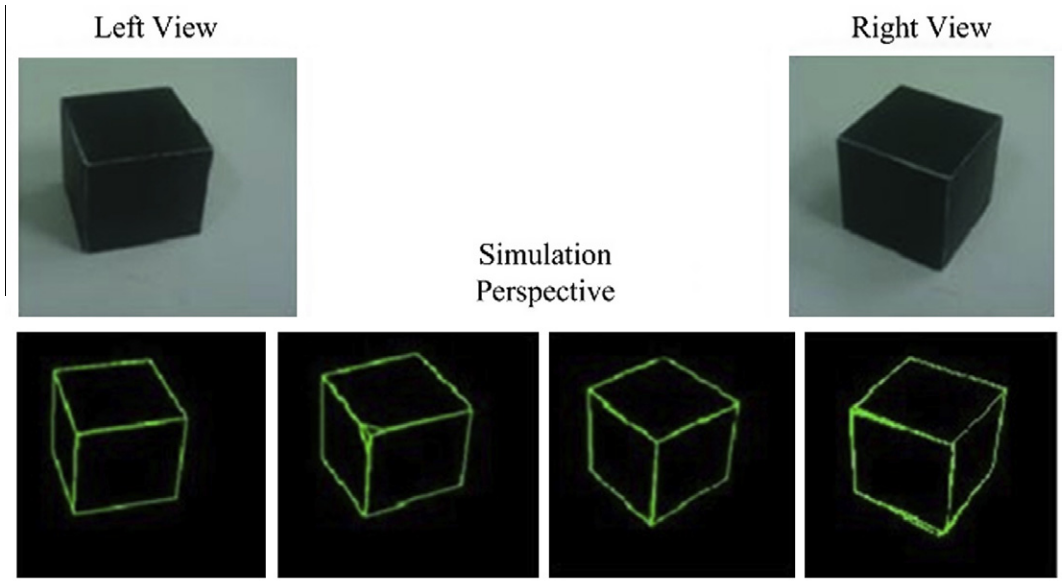

Fig. 9. Input images and simulated images.

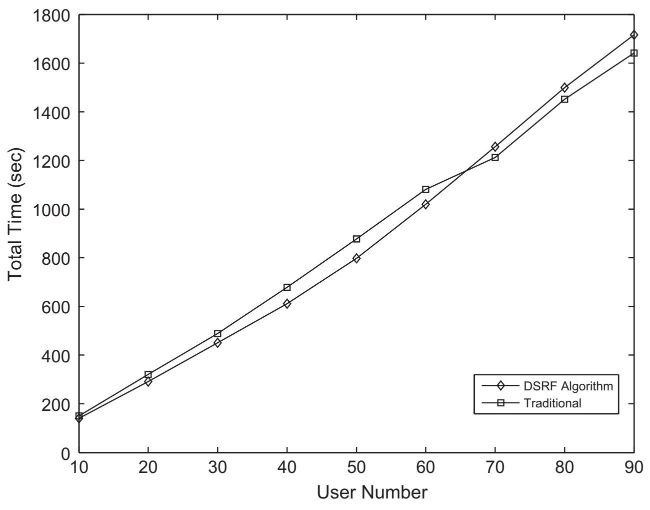

Fig. 10. The influence of user number to the total time.

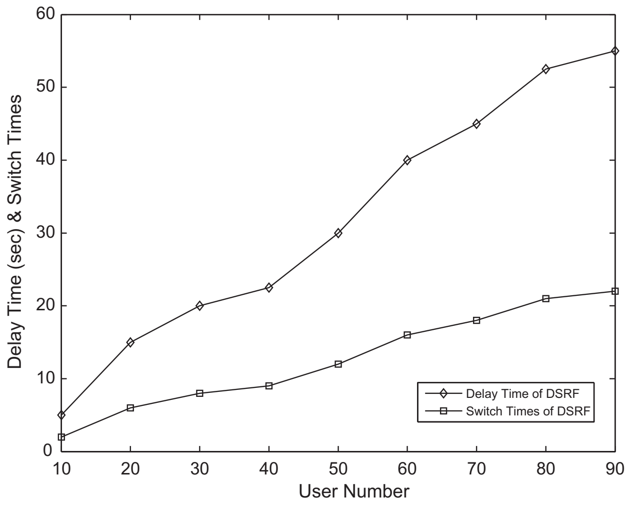

Fig. 12. The influcence of user number to delay time and switch times of a combination task.

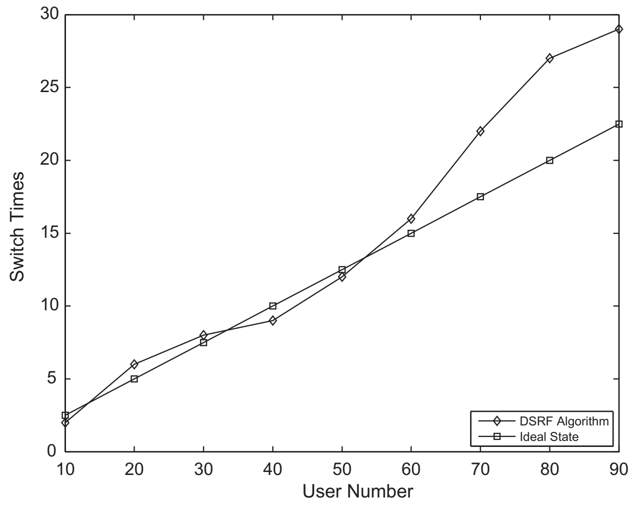

Fig. 11. The influence of user number to switch times of DSRF algorithm.

mance of DRSF algorithm and the throughput of DSRF algorithm gradually gets lower than Hadoop native.

Fig. 14 displays the average waiting time of each user when the user number increases. The waiting time means the time from the user requests the image to the user receives the image. In the figure, when the user number exceeds 60, our proposed DSRF algorithm uses most of the waiting time in switching tasks. Nevertheless, the figure further shows that when the user number is under 70, the waiting time of our DSRF algorithm is obviously less than traditional Hadoop MapReduce.

Fig. 15 compares the maximal queue number of traditional Hadoop MapReduce and our proposed DSRF algorithm when the user number is 40. The figure shows that with less user number, the queue number of Hadoop Native and DSRF algorithm is very close. Thus, we can know that Hadoop Native and DSRF algorithm result to similar system stability but DSRF in fact performs better.

Fig. 16 compares the maximal queue number when the user number is 80. Our proposed DSRF algorithm is able to maintain the queue number within a stable range. However, with the

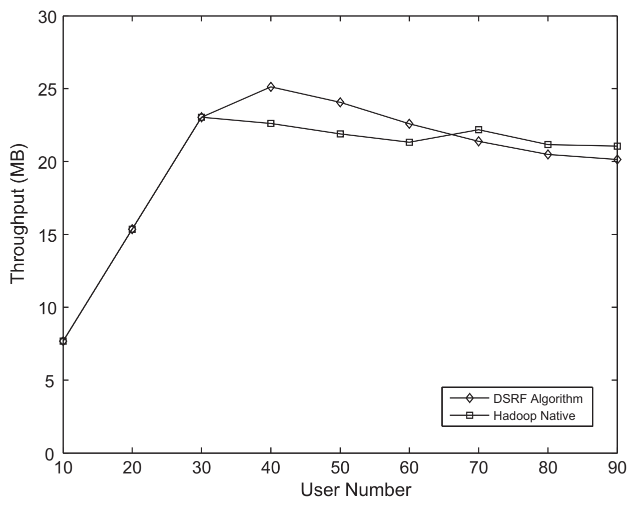

Fig. 13. The influence of user number to the throughput.

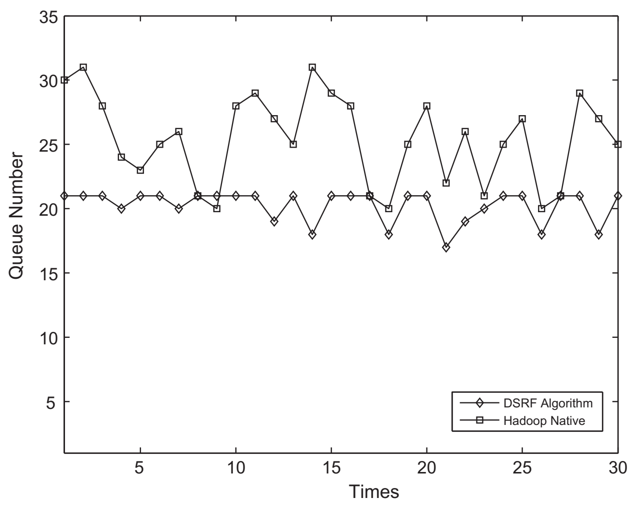

Fig. 16. Relation between user number and the maximal queue number (user number = 80).

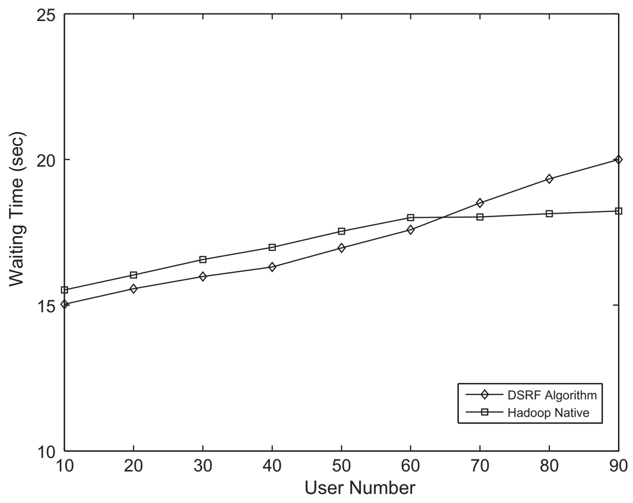

Fig. 14. The influence of user number to the average waiting time.

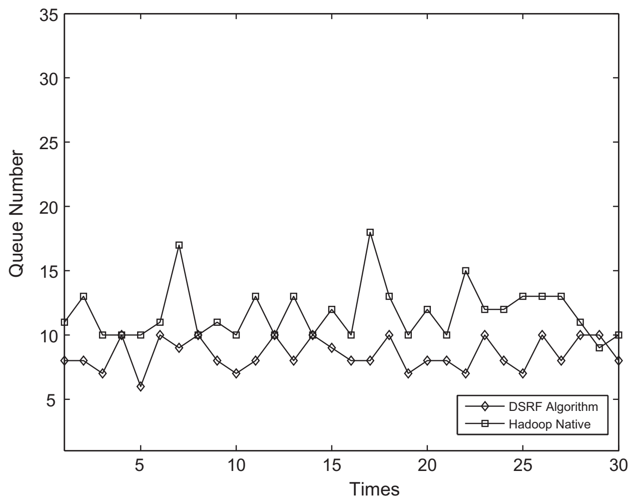

Fig. 15. Relation between user number and the maximal queue number (user number = 40).

increase of user number, Hadoop Native results to severe vibration of the maximal queue number. Therefore, it reveals that traditional Hadoop Native is not as stable as DSRF algorithm.

5. Conclusion

MapReduce is presented to process vast amounts of data and to return the result to users within the minimum time. Most users use MapReduce to handle applications of higher speed and lower computing complexity. In this paper, we implement MapReduce on an integrated 2D to 3D multi-user system, in which the Map function is responsible for image processing procedures with high complexity and high computation and the Reduce function is responsible for combining the intermediate data processed by the Map function and creating the final output. Applications of high complexity and high computation will prolong the execution of the Map function and delay the following tasks. Thus, this paper presents a DSRF (Dynamic Switch of Reduce Function) algorithm to switch to different tasks dynamically according to the achieved percentage of tasks. When the waiting time increases with the user number, our DSRF algorithm allows the Reduce function to utilize the waiting time in computing other tasks. In this way, not only the waiting time and the computing time can be reduced, but also users can get the image results quickly and the MapReduce system can reach higher performance. Our future target is to include more quality schemes, like QoE and QoS, so that better quality can be guaranteed in the cloud infrastructure to enhance cloud-based transmissions.

References

[1] Q. He, Z. Li, X. Zhang, Study on cloud storage system based on distributed storage systems, in: Proc. International Conference on Computational and Information Sciences (ICCIS), Dec. 17–19, 2010, pp. 1332–1335.

[2] J. Jiang, Y. Wu, X. Huang, G. Yang, W. Zheng, Online video playing on smartphones: a context-aware approach based on cloud computing, Journal of Internet Technology 11 (6) (2010) 821–828.

[3] C.-H. Chiu, H.-T. Lin, S.-M. Yuan, CloudEdge: a content delivery system for storage service in cloud environment, International Journal of Ad Hoc and Ubiquitous Computing 6 (4) (2010) 252–262.

[4] Y. Xu, J. Yan, A cloud-based design of smart car information services, Journal of Internet Technology 13 (2) (2012) 317–326.

[5] Y.-X. Lai, C.-F. Lai, C.-C. Hu, H.-C. Chao, Y.-M. Huang, A personalized mobile IPTV system with seamless video reconstruction algorithm in cloud networks, International Journal of Communication Systems 24 (10) (2011) 1375–1387.

[6] K.D. Bowers, A. Juels, A. Oprea, HAIL: a high-availability and integrity layer for cloud storage, in: Proc. the 16th ACM conference on Computer and communications security (CCS '09), 2009, pp. 187–198.
[7] M. Luo, G. Liu, Distributed log information processing with Map-Reduce: a case study from raw data to final models, in: Proc. IEEE International Conference on Information Theory and Information Security (ICITIS), Dec. 17–19, 2010, pp. 1143–1146.
[8] F.N. Afrati, J.D. Ullman, Optimizing multiway joins in a Map-Reduce environment, IEEE Transactions on Knowledge and Data Engineering 23 (9) (2011) 1282–1298.
[9] C. Zhang, H.D. Sterck, CloudBATCH: a batch job queuing system on clouds with Hadoop and HBase, in: Proc. IEEE Second International Conference on Cloud Computing Technology and Science, 2010, pp. 368–375.
[10] J. Polo, D. Carrera, Y. Becerra, M. Steinder, I. Whalley, Performance-driven task co-scheduling for mapreduce environments, in: Proc. IEEE Network Operations and Management Symposium (NOMS), 2010, pp. 373–380.
[11] T. Sandholm, K. Lai, Dynamic proportional share scheduling in Hadoop, 15th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '10), 2010.
[12] J. Yu, R. Buyya, A budget constrained scheduling of workflow applications on utility grids using genetic algorithms, in: Workshop on Workflows in Support of Large-Scale Science (WORKS '06), 2006.
[13] A. Sakatoku, T. Osada, G. Kitagata, N. Shiratori, T. Kinoshita, 3D symbiotic environment for agent-aided collaborative work, Journal of Internet Technology 13 (1) (2012) 127–136.
[14] M. Zaharia, D. Borthakur, J.S. Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling, in: Proc. the Fifth European conference on Computer systems (EuroSys '10), NY, USA, 2010, pp. 265–278.
[15] J. Ekanayake, S. Pallickara, G. Fox, MapReduce for data intensive scientific analyses, in: Proc. IEEE Fourth International Conference on eScience (eScience '08), 2008, pp. 277–284.
[16] K.-R. Lee, M.-H. Fu, Y.-H. Kuo, A hierarchical scheduling strategy for the composition services architecture based on cloud computing, in: Proc. the Second International Conference on Next Generation Information Technology (ICNIT), Jun. 21–23, 2011, pp. 163–169.
[17] T. Gunarathne, T.-L. Wu, J. Qiu, G. Fox, Cloud computing paradigms for pleasingly parallel biomedical applications, in: Proc. the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10), New York, USA, 2010, pp. 460–469.
[18] J. Ekanayake, T. Gunarathne, J. Qiu, Cloud technologies for bioinformatics applications, IEEE Transactions on Parallel and Distributed Systems 22 (6) (2011) 998–1011.