Journal of Software Engineering and Applications, 2015, 8, 395-406
Published Online August 2015 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2015.88039

The Optimization and Improvement of MapReduce in Web Data Mining

Jun Qu, Chang-Qing Yin, Shangwei Song

[1]Logistics Department, Tongji University, Shanghai, China
[2]School of Software Engineering, Tongji University, Shanghai, China
[3]College of Design and Innovation, Tongji University, Shanghai, China
Email: yinchangqing@tongji.edu.cn

Received 4 July 2015; accepted 21 August 2015; published 24 August 2015

Abstract

Extracting and mining social networks information from massive Web data is of both theoretical and practical significance. However, one of definite features of this task was a large scale data processing, which remained to be a great challenge that would be addressed. MapReduce is a kind of distributed programming model. Just through the implementation of map and reduce those two functions, the distributed tasks can work well. Nevertheless, this model does not directly support heterogeneous datasets processing, while heterogeneous datasets are common in Web. This article proposes a new framework which improves original MapReduce framework into a new one called Map-Reduce-Merge. It adds merge phase that can efficiently solve the problems of heterogeneous data processing. At the same time, some works of optimization and improvement are done based on the features of Web data.

Keywords

Cloud Computing, Web Data, MapReduce, Map-Reduce-Merge

1. Introduction

Social network consists of several categories of social entities and the relationship among them. Social Network Analysis (SNA) is an important tool to understand the behavior of human and analyze the social architecture. Social Network Analysis can not only be applied in sociology, but also in the informatics, information retrieval, information behavior and information metrology. In addition, Social Network Analysis has a significant effect on network knowledge mining, scientific evaluation, network information behavior research and knowledge management.

However, the theoretical and practical value of SNA depends on the quality and reliability of social network data. If the data itself is low in quality, the analysis result from that would be nonsense. Traditional sociology mainly acquires data via social investigation and group sampling. As the research in social network analysis develops, traditional method for acquiring social network data cannot meet the requirement of large scale social network analysis in many aspects. In the recent years, the appearance of various web platforms with social feature, such as online forum, twitter, SNS, makes it possible to extract and mining large quantity of high reliable and quality social network information from huge web data source via computer.

Web data sources usually contain massive entity, the number of relationships between these entities is the square of the number of entities, and massive computing provides challenges for research on this issue. In recent years, with the rapid growth of information and data in the age of the Internet, the concept of cloud computing was proposed. Cloud computing is an emerging model of business computing which is the further development of distributed computing, parallel processing and grid computing. It is able to provide a wide variety of Internet applications hardware services, infrastructure services, platform services, software services and storage service system. The sophistication of cloud computing platform provides new parallel computing framework for massive data, especially for Web data source large-scale data acquisition and an effective way for massive scale data processing.

But how to implement a Web mining based on the core computing model MapReduce of cloud computing is still a problem to be solved. As we all know, MapReduce is efficient when deals with isomorphic data, but the performance faced with heterogeneous data is often less than ideal. However, there are always heterogeneous data sets with no fixed format in the Web. In order to solve this problem in the existing MapReduce framework, developers need to write additional code, which is not what we expect.

In Web data mining, processing relational data is very common, especially for the user characteristic extraction and analysis. Like Facebook and Twitter have a large amount of user information to do extraction and mining these two sites combined information would be better than process only one site, so we need to do an effective integration on the heterogeneous information obtained from the two sites.

In this paper, the general concepts and definitions of MapReduce are provided, and introduce an improved model in Web data mining named Map-Reduce-Merge, merge the heterogeneous data produced by Reduce end effectively by increasing the Merge stage. In the meantime, it enhances the efficiency of Web data mining through optimizing the scheduling strategy, Map and Reduce tasks.

2. Map-Reduce

2.1. Overview

MapReduce is a programming model proposed by Google [1], it combines file system GFS [2] to implement parallel computing for large data set on massive scale distributed servers system. The concepts of "Map" and "Reduce", and their basic thoughts come from the features of functional program language and vector program language. MapReduce makes it greatly available for developers to deploy their programs on distributed system without knowledge of parallel programming.

MapReduce model consists of map function and reduce function. After input data processed by map function, it will produce a local intermediate result of key/value pairs. Reduce function remotely use the key/value pairs produced by map function as input to process under the schedule of Master. In order to take the independence and correlation of distributed data processing into consideration, it will then combine the result with same key value to get the final output. Doug Cutting used Java developed open source project Hadoop to implement MapReduce mechanism [3], and HDFS distributed file system [4], which allows the enterprises all over the world have a chance to utilize MapReduce to implement large scale distributed data processing.

MapReduce is not simply divided into Map and Reduce operations. On the one hand it has a more detailed division of operation, such as Combine, Shuffle and Sort. On the other hand in order to achieve large-scale data parallel and distributed processing it does a compact capsulation. The whole architecture help users to finish much hard work and solve problems like data partitioning, scheduling, data and code co-located, the process synchronous communication, fault tolerance and failure handling, load balancing and other issues [5] and make these functions transparent to the developers. Developers only need to implement the Map and Reduce interface, without concentrating on the underlying system-level problems, to complete the development of parallel programs on distributed clusters.

Certainly MapReduce has its own drawbacks [5].

1) High start consumption, when starting the cluster it visited all nodes in the network.

2) Low efficiency of random disk access. The MapReduce distributed file system is sequentially read and blocks access.

3) Synchronization mechanism is the hardest problem, the MapReduce tasks do not support the sharing of global data.

Besides, Mappers and Reducers run independently, they cannot interact through some other mechanism. But under the background of large-scale data, these disadvantages are related to applications and tolerant compared with their advantages. And they can also be alleviated or eliminated through a variety of optimization.

2.2. Programming Model

MapReduce is a simple programming model for data processing. A MapReduce program can be applied to a data set, which means a job. One job generally consists of several or even hundreds of tasks. The control machine responsible for assigning tasks in MapReduce is called master, the machine for executing task is called worker (core multi-thread, multi-core, multi-processor can also be regarded as worker). They are called master and slave in Hadoop respectively. From the view of storage they are divided into NameNode and DataNode. Namenode recorded the location and status information of distributed file block storage, whereas DataNode is responsible for the storage of real data, and under normal circumstances, DataNode is also the Task executor. Under the default setting of MapReduce, files are stored as blocks in a distributed file system [6].

The standard workflow of MapReduce is shown as **Figure 1**.

1) When a job is submitted, according to the distribution of file blocks in distributed systems, jobs are divided into several sub-tasks and processed by Mappers (the worker execute Map tasks). Generally tasks will be assigned to the machine contains data or the machine in the same rack to improve the processing speed, which is so-called "code find data" mode.

2) Each Mapper executes on different file block, according to the Map execution program, to transform data into key/value pairs. And as for each key/value pair, it executes the Map function provided by users to process. This stage is the massive parallel execution stage.

3) When Map task is completed, there will be Shuffle and Sort stage in the framework. It will distribute and sort the data produced by the Mappers and write them into local file system for the next stage, which improve the efficiency of Reduce.

4) Reduce tasks obtain their own data from the output of Map tasks, download to the local and merge them.

The mapping relationship of key/value can be illustrated by the following two formulas.

$$\text{map}\left(\text{k1, v1}\right) \to \text{list}\left(\text{k2, v2}\right)$$

$$\text{reduce}\left(\text{k2, list}\left(\text{v2}\right)\right) \to \text{list}\left(\text{v2}\right)$$

In the entire process, the user can assure the framework running by only implementing Map and Reduce functions, Which means only the two functions used in 2) 4) stages require the user to specify, whereas other stages are completed by the framework.

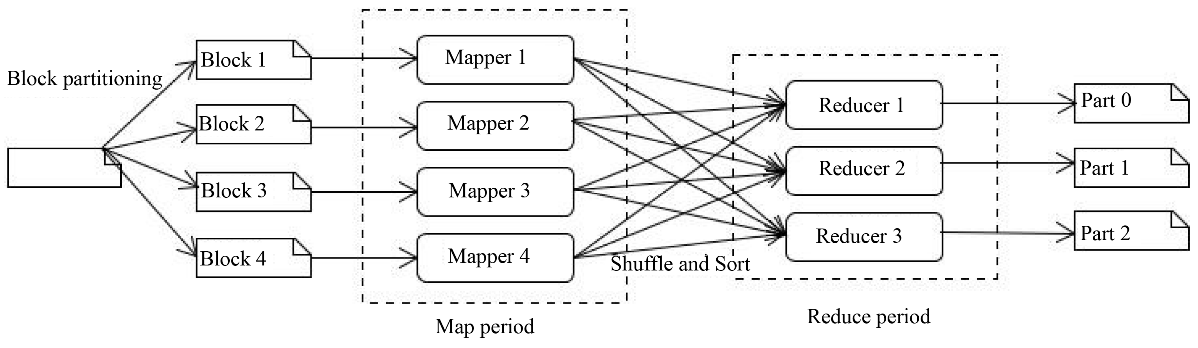


Figure 1. The standard workflow of MapReduce.

2.3. Schedule Execution Process

In Hadoop, there are two kinds of nodes to control the execution process of jobs: one JobTracker and several TaskTracker. JobTracker executes tasks by schedule TaskTracker, to coordinate all jobs on the platform. TaskTracker execute tasks and report the execution progress to JobTracker. As a result, every job has all corresponding records in a execution process. If any task fails, JobTracker will schedule another TaskTraker to execute the same task [7].

The job is initially generated in the user node, which communicate (RPC) through JobClient object and JobTracker. At first, ID of the job is got from the JobTracker. Then, job resources are submitted, including job data, job configuration information and MapReduce applications to the shared file system HDFS. Finally, job is submitted to JobTracker. The JobTracker firstly initialize the job, and then divide the data into the input split. The process is performed at logic level, rather than the actual data manipulation. TaskTracker periodically sent heartbeat to the JobTracker for the task. According to their own scheduling policy, JobTracker selects the appropriate task for TaskTracker and return to the TaskTracker by heartbeat. TaskTracker launches a separate virtual machine and perform task for an individual Map or Reduce task [8].

3. Map-Reduce-Merge

3.1. Programming Model

Map-Reduce-Merge model can handle multiple heterogeneous data sets, and its basic characteristics are as follows (α, β, γ represent different data set, k represent key and v represent value entity):

In this model, the map function will convert a key/value input (k1, v1) to an intermediate key/value pairs [(k2, v2)]. Reduce function will make value of [v2] whose key is k2 gathered together, to produce a value of [v3], which is associated with k2. Noted here, the input and output of the two functions are in the same data set α. Another pair of map and reduce functions produce intermediate output (K3 [v4]) from another dataset beta. Based on the keys: K2 and K3, merge function can merged into a key/value results (k4, v5) from the two output of Reduced function by different data sets. This ultimately results generate a new data set γ (**Figure 2**). If $\alpha = \beta$, then the merge function will do a self-merge, similar to the self-join in relational algebra.

The characteristics of Map and Reduce in new model are almost the same as its original MapReduce. The only difference is that here Reduce output is a key/value pairs, not just values. This change is introduced because the merge function needs the input of data set consisting of key/value. In the Google MapReduce, the result of Reduce is the final result, users can package any required data into [V3], and there is no need to pas k2 to the next stage.

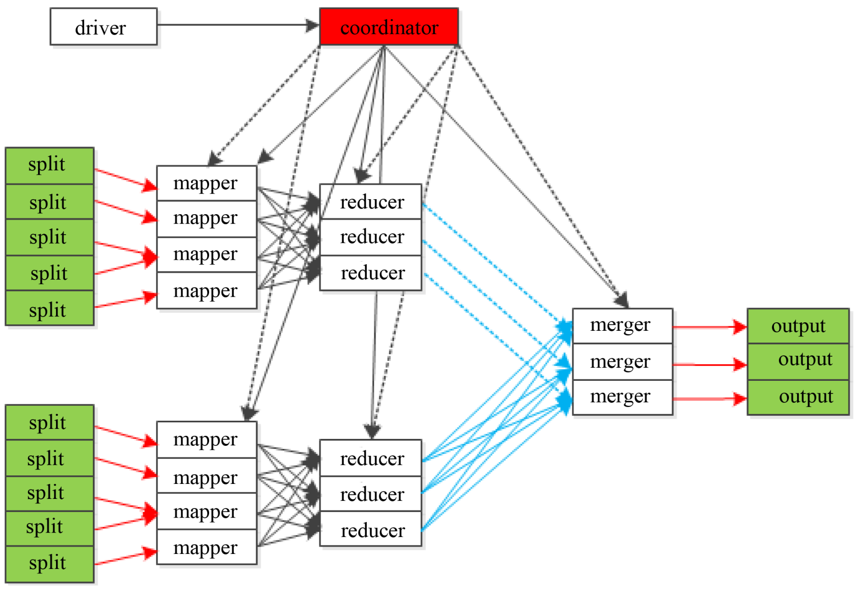


Figure 2. Map-Reduce-Merge model.

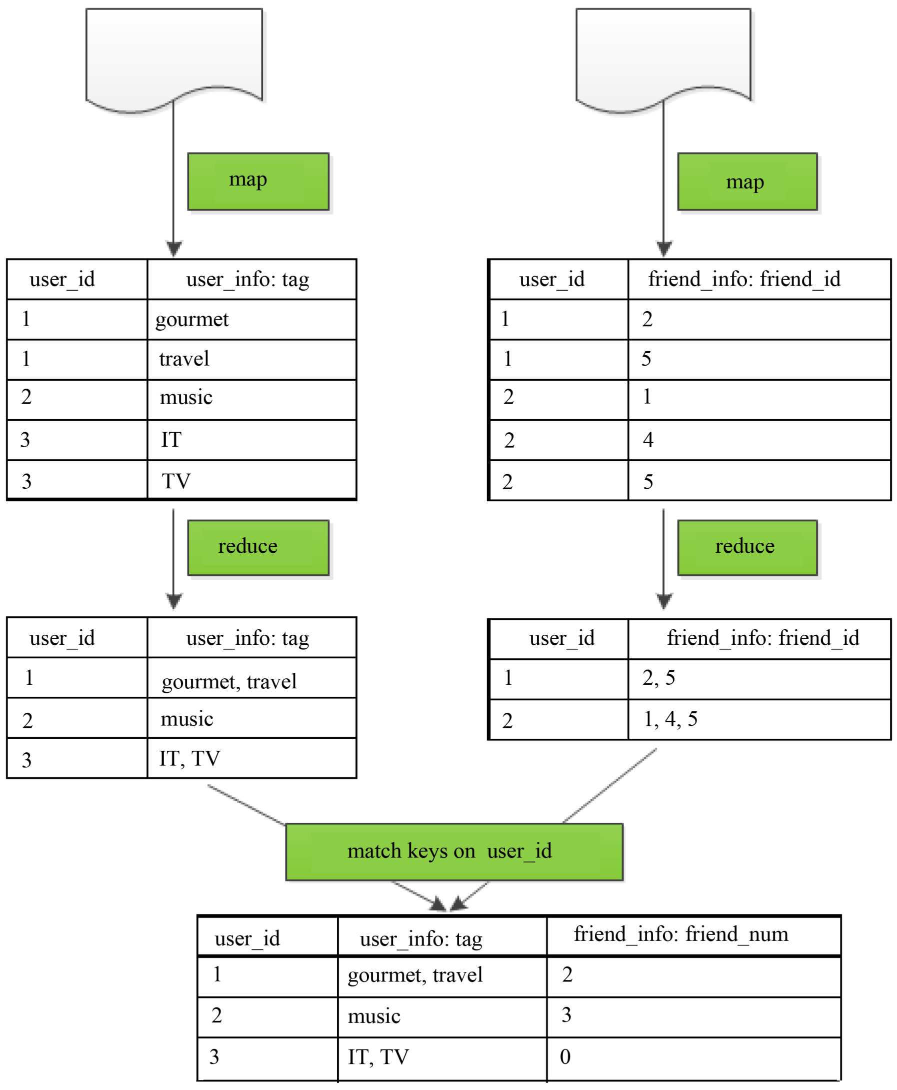
In order to build the merge function to read data from multiple data sets, design emphasizes to pass the key: k2 from the Map to Reduce, and then to the merge function. This can ensure that the data is divided into the areas and classified by keys before the Merge.

3.2. Example of SinaWeibo

In this section, a simple example of SinaWeibo will be proposed to illustrate how Map, Reduce and Merge module work together. Here we have two data sets, User and Friend Relationship. The key property of User is user_id, other information is included in user_info "value". The key property of Friend Relationship is user_id, other information is included in friend_info "value". Here the data processing is a combination of the two data sets and calculates the number of Weibo users "followers".

The left part of **Figure 3** is the table of every entity tag produced by Mapper processing User entity. Then Reducer merges the tag of every user and classifies them based on user_id. The right part is the table of every entity tag produced by Mapper processing Friend Relationship entity. Then Reducer merges the tag of every user and classifies them based on user_id. Finally Merge obtains output from two Renders and merge based on user_id and algorithm.

The code of Mappers and Reducers are as follows:



user_id	user_info: tag
1	gourmet
1	travel
2	music
3	IT
3	TV

user_id	friend_info: friend_id
1	2
1	5
2	1
2	4
2	5

user_id	user_info: tag
1	gourmet, travel
2	music
3	IT, TV

user_id	friend_info: friend_id
1	2, 5
2	1, 4, 5

user_id	user_info: tag	friend_info: friend_num
1	gourmet, travel	2
2	music	3
3	IT, TV	0

Figure 3. Combine and calculate number of "followers".

```
/* Algorithm 1 Map function for the User dataset. */
Map(const Key& key,/* user id */
    const Value& value /* user info */){
    user_id= key;
    tag=value.tag;
    /* compute tag using user info */
    output key =(user_id);
    output value =(tag);
    Emit(output key, output value);
}
/* Algorithm 2 Map function for the FriendRelationship dataset. */
Map(const Key& key,/* user id */
const Value& value /* firend info */){
user_id= key;
friend_id=value.firend_info:user_id;
Emit((user_id),(friend_id));
}
/* Algorithm 3 Reduce function for the User dataset. */
Reduce(const Key& key,/* user id */
    constValueIterator& value
    /* an iterator for a tags collection */){
    tag sum =/* sum up tags for each user id */
    Emit(key,(tag sum));
}
/* Algorithm 4 Reduce function for the FriendRelationship dataset. */
Reduce(const Key& key,/* (user id) */
constValueIterator& value
   /* an iterator on a friends collection */) {
   /* aggregate friends and compute friend_num */
Emit(key, (friend_num));
}
```

3.3. Model Implementation

So far has already implemented a Map-Reduce-Merge framework, whose Map and Reduce module has some trivial changes from MapReduce of Google. Merge is similar to Map and Reduce, developers can implement the user-defined logic of data processing. Calling mapper will produce one key/value pair. Calling Reducer will produce a set of value classified by key. Merger will process two key/value pairs which come from different sources.

At the Merge stage, users would adopt different data processing logic according to data resource. After the mission of Map and Reduce is completed, the Coordinator of Map-Reduce-Merge will start Mergers on a cluster of Nodes. When Merger starts, a Merger ID will be assigned to it. With this ID, Partition Selector can decide from which Render can Merge get input data. Similarly, Mappers and Reducers are also assigned an ID. As for Mappers, this ID stands for Input File Split. As for Reducers, this ID stands for Input Bucket. Mappers split and store their output. For the users of MapReduce, there IDs is the detail of system implementation. In Map-Reduce-Merge, users associate these IDs with the output and input between Mergers and Reducers.

4. Optimization

Aim at the features of Web data extraction and mining, such as small single data, large total amount of data, the existence of network latency, this chapter will provide some mechanism about model optimization, especially for the new Merge stage, it will provide specialized strategies to reduce consumption of resources (like number of network connection and disk bandwidth).

4.1. Schedule Strategy

The scheduler is the decision-makers which is running on the JobTracker and responsible for scheduling the jobs submitted by users. Users will submit jobs to the job queue to wait to be scheduled, the scheduler firstly select the job in the job queue and initialize it, then depending on the scheduling strategy, assign the tasks included in the job to the slot on TaskTracker to execute [9] (TaskTracker will periodically send heartbeat request to JobTracker to acquire tasks).

Because schedule strategy has significant impact on the efficient use of the cluster, load balancing, job execution efficiency [10], it has a great significance to provide an efficient scheduling execution environment to MapReduce-Merge programming model framework. Recently, the most influential scheduling strategies include the Capacity scheduler, Fair scheduler and LATE scheduler. They make improvements for the original FIFO scheduling algorithm, parallel scheduling in multi-user management, and cluster support for heterogeneous environments respectively.

FIFO scheduler is a Hadoop default task scheduler, it simply schedule the jobs submitted by users via a job queue. But it has a poor performance for multi-user and for the efficient use of the cluster.

Capacity scheduler can support multiple organizations to share the same large clusters, and ensure that each organization have the smallest computing power. But Capacity scheduler allocate resources equally, tend to regard the resources of jobs submitted by users is relatively equally, it does not take the diversity of job requirements into consideration.

Fair scheduling ensure small job has response time as short as possible through fair resource scheduling strategy, and provides service level assurance for product operations [11]. However as for the data analysis job which consume large computing resource, its prior schedule will result in life cycle delay of up to a task. This will have a great influence on the subsequent execution of jobs.

The LATE scheduler is the default scheduler in a heterogeneous environment which has a “try to execute” improvement on strategy [10]. But the determination of task execution time is based on the premise that all tasks are carried out in accordance with the constant rate of execution. With the change of use of joint resources, such as memory consumption, the rate of task execution is not constant. In addition, the idea of Reduce divide the execution process into 3 equal parts for copy, merge and reduce is not accurate.

Can be found, for the special case of Web data, there are certain limitation for existing scheduler. In order to better fulfill the requirement of Web data, the schedule strategy should be improved as follows:

1) Using the multi-queue to process jobs submitted by users. Like Capacity scheduler, every queue correspond to a user group, administrator can manage the users and their queue. Queue has priority according to that of user group. The group of the queue can submit jobs with different priority, which correspond to the urgency of the task, such as real-time operating can have higher job priority data whereas analysis jobs correspond to a lower priority, multi-queue scheduling support preemption.

2) Classify the type of tasks. Job type is the result of refinement for requirement of job resource. So far job type is divided into CPU-intensive and disk-intensive which is similar to three queue scheduler. Allocate a certain amount of computing resource for different job type and compute parallel can enhance the utilization of cluster. The multi-type of job scheduling policy is a best-effort scheduling under the premise of ensuring the multi-queue scheduling. Finally, introduce competition mechanism for resources among multi-user queue.

4.2. Map

In the process of Map task, the intermediate key/value pair <k2, v2> outputted by map function is written to a circular cache in logic, rather than immediately written to the local file system. When the circular cache is full, it will continue to receive the output from map function while write intermediate file to disk at the same time [12].

The output file is as follows: do partition and sort operation to the key/value pair in cache. Then spill file to disk.

If there are much the intermediate data in a Map task, it need spill several times to disk. Every Spill will get a file, so multiple operations will result in multiple file which is after segmentation and sort. At the end of Map task execution, it will merge all spill file. Under the guarantee of the order of split and sort is unchanged, merge all the intermediate files into one file.

Obviously this intermediate file organization is not compact enough, so it can be optimized. Increase one thread to merge the output of multiple Map tasks. This thread is started by the TaskTracker and will periodically check the number of Map output files of each task on the node. When the number of map output files of a certain task reaches the default size of configuration, the files will be merged. The file merge is the same way as the merge of multiple spill file, which is a merge on the disk to save the memory consumption of the node.

By this merge operation, reduce the number of intermediate files can be reduced, which makes the output data of the same the Map task on a node more compact. This compact organizational structure has two advantages:

1) The less output file can reduce the overhead of establishing a network connection in the Reduce task Shuffle stage, to improve the speed of data transmission on the network.

2) The merge operation of Map end reduces the workload of the Reduce task merge phase. In the download and merge stage of Reduce task, it will merge the Map intermediate data several times until get a complete Reduce input and then call reduce function. Merge in the node contains Map can reduce the number of intermediate files , and then reduce the round of merge, finally reduce the execution time.

4.3. Reduce

Reduce end can be improved in the following two aspects.

1) Unbalance of Reduce tasks

The reasons for unbalanced data can be summarized into two categories [13] [14]:

a) Intermediate results are key dispersed, while after partition too is aggregation. Although the number of different key is large, but after mapping, too many key gathered in the same Reduce task.

b) Intermediate results are key single, while result is diverse. The type of key number is relatively small, but the number of records corresponding to the key is large. According to the principle that the records with same key is mapping to the same Reduce, one Reduce can process all the records of one key.

Aim at the unbalance in the data, the improvement target is to make the data be evenly distributed to each Reduce task to ensure that each amount of Reduce task can be roughly equal. To avoid some nodes to deal with too much data "exhausting", while others node have no data processing "starve to death".

Improvement program will start twice MapReduce Job. Firstly, use the Reduce in first MapReduce to merge locally, to ensure Reduce operation data in the second reduce operation is balanced and the records with same key can be processed in the same Reduce task.

For the first input of the Reduce stage, it is not necessary to ensure that each key corresponding to the record must be mapped to the same Reduce. On the contrary, each record should be randomly mapped to a reduce records having the same key which is uniformly dispersed, but need to ensure that the amount of calculation in each Reduce should be substantially equal. Reduce operating merged the records with same key record into one, after the local polymerization data is greatly reduced, and there is only one record corresponding to one key in each Reduce output. At this point the data as the second input of MapReduce, map function have nothing to do, directly regard the input data as output, the data is mapped to Reduce task in accordance with the original partition. Reduce do one operation, the same data is combined to give the final output of each key corresponding to a result.

Figure 4 shows the data processing flow for improved Map-Reduce-Merge.

However this does not mean that all the calculations need local polymerization, if Map output can reach Reduce equilibrium requirements, this program will be executed. So need to develop a strategy to determine under what circumstances local data aggregation should be operated. At the beginning of Map firstly attempts to do Hash mapping, if the mapping of the different number of records/Try to certain records the number of records is greater than a given threshold, then it means the Map output data and input to the Reduce task is balanced, so just execute the original MapReduce process. Otherwise launch another MapReduce Job and operate local polymerization in the first Reduce.

2) The I/O problem in shuffle phase of Reduce

During the execution of the Reduce task, the Shuffle stage of I/O is often a performance bottleneck. In this stage, Reduce task will periodically inquiry the Map job information completed by JobTracker. For the completed Map task, the download threads will establish an HTTP connection between its nodes and download the intermediate data. When the output of the Map task download is complete, the connection is disconnected. Therefore, for a Reduce task in the Shuffle stage, at least creates the download connection with the same number of the Map tasks.

The average overhead for establishment of connection in less complex cluster of a network topology is the tens to hundreds of milliseconds, which is very short. But for the relatively small amount of output data of each Map, the overhead of establishment cannot be underestimated. Especially for Web data, this situation is particularly evident, most of the Map output is obtained from each of the sub-page jobs with small the amount of data.

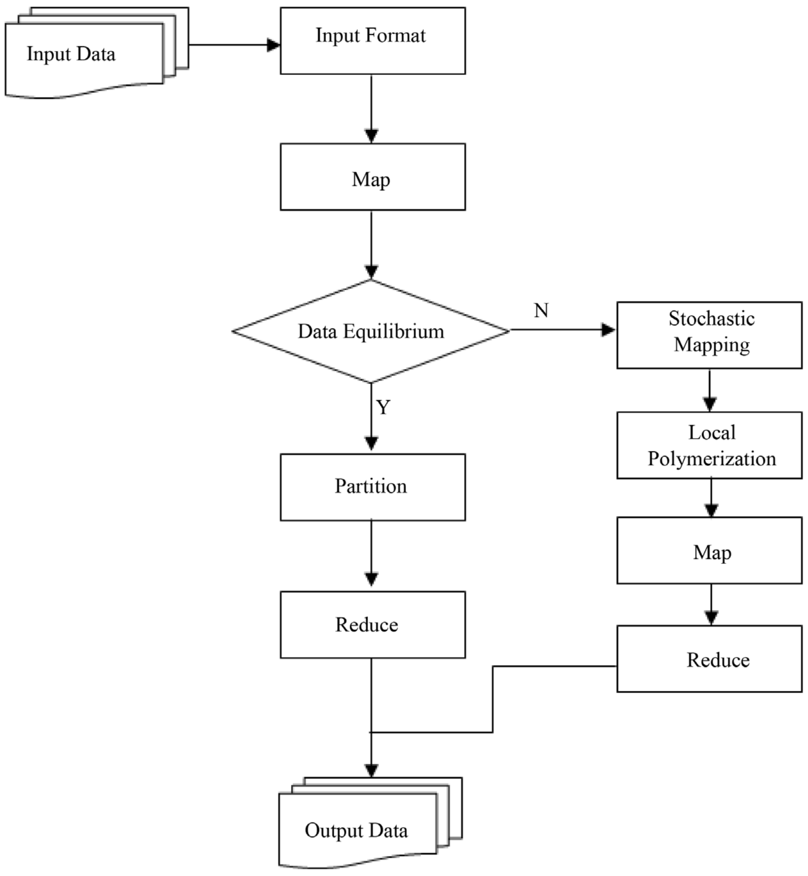


Figure 4. Improved Map-Reduce-Merge flowchart.

To optimize I/O in shuffle phase, firstly reduce the number of connections to establish, for each download thread in its life cycle, only to establish a long connection to the same node. When the output of the MAP can be downloaded, then download it from this link, after all output complete downloading, then close the connection. This is effective for jobs with many maps.

Because http is based on the TCP connection to transfer files, TCP slow start-up characteristics determine the data transfer rate cannot immediately reach the network bandwidth. The proposed method to merge Map output file can increase the amount of once data transmitted, which can improve the transmission rate [15].

Map output merger changed the organizational form of the intermediate data. Shuffle download thread will not necessarily be able to directly download a Map output file according to file information get from JobTracker It may get the output file, which may be obtained by combined Map output file. In this regard, solution is to keep the original Map task progress reporting mode and Shuffle stage download access to information the same way. Only make minor modifications in the implementation process of the download, as far as possible through the merger of the intermediate output file, making a large amount of data transferred.

For example, the map A task execution is complete and report to the JobTracker. Then Reduce tasks query to JobTracker for the completion of Map A, and try to communicate to the node contains map A and download the output data of Map A. At the same time, Map B is completed in the same node as Map A, and the output of Map A and Map B has been merged into an output file. The node will send combined output to the Reduce node, and inform that the transferred data is a result of the merger of A and B tasks. After Reduce tasks have received the data, it will mark Map A and MAP B as received, and do not further access to the node for the output data of the task B.

For a special case, the completion information of Map A and Map B has been acquired by Reduce separately, downloaded by two download threads. The nodes will receive one of the download requests, and send the merge of A and B. Reduce take retreat, does not immediately re-attempt when the download request of connection to the same node is refused, the download will not restart until all the threads connected to the node are completed and not able to obtain the desired output data .

4.4. Workflow

The MapReduce program strictly abides by the two-stage process, the first Map and the second Reduce. The user can change the default configuration. However, some basic operations, such as Partition and Sort are built-in, and must be executed. It is a little troublesome for the users just want to perform Map or Reduce tasks. Although such restrictions make MapReduce simple, but it cannot meet the needs of advanced users, who more often want to customize the entire workflow. So it should be possible to optimize the interface of the framework to allow advanced users to have greater freedom to define the workflow to meet their own requirements.

Because MapReduce has only two stages, the customization is relatively simple. After add Merge stage, there can be a combination of more kinds of processes to meet the specific data processing tasks (see **Figure 5**).

The left is the typical 2-pass MapReduce workflow. The entire process only contains a Map and Reduce.

The middle is the 3-pass Map-Reduce-Merge workflow. The whole process consists of two Map and Reduce a Merge.

The right is a multi-pass hierarchical workflow. The entire process contains multiple Map Reduce and merge.

4.5. Load Balancing

Load balancing helps to evenly dispersed load to the junction point of the idle when the load exceeds the threshold level in a junction point. Even though load balancing is not obvious enough when executing Map-Reduce-Merge algorithm, however, in processing large file, and hardware resources [16] utilization is critical the advantage is very obvious. A significant role in the tight resource situation is to increase hardware utilization, improve performance. To balance some of the data node which is full or new, empty node joins the cluster, implement a module to balance disk space usage on the cluster of distributed file system. If for each data point, the junction space and the ratio of the total capacity (Junction Point utilization) is different from the used space and total space ratio (Clusters utilization) on the cluster and does not exceed the threshold value, then the cluster is regarded as balanced.

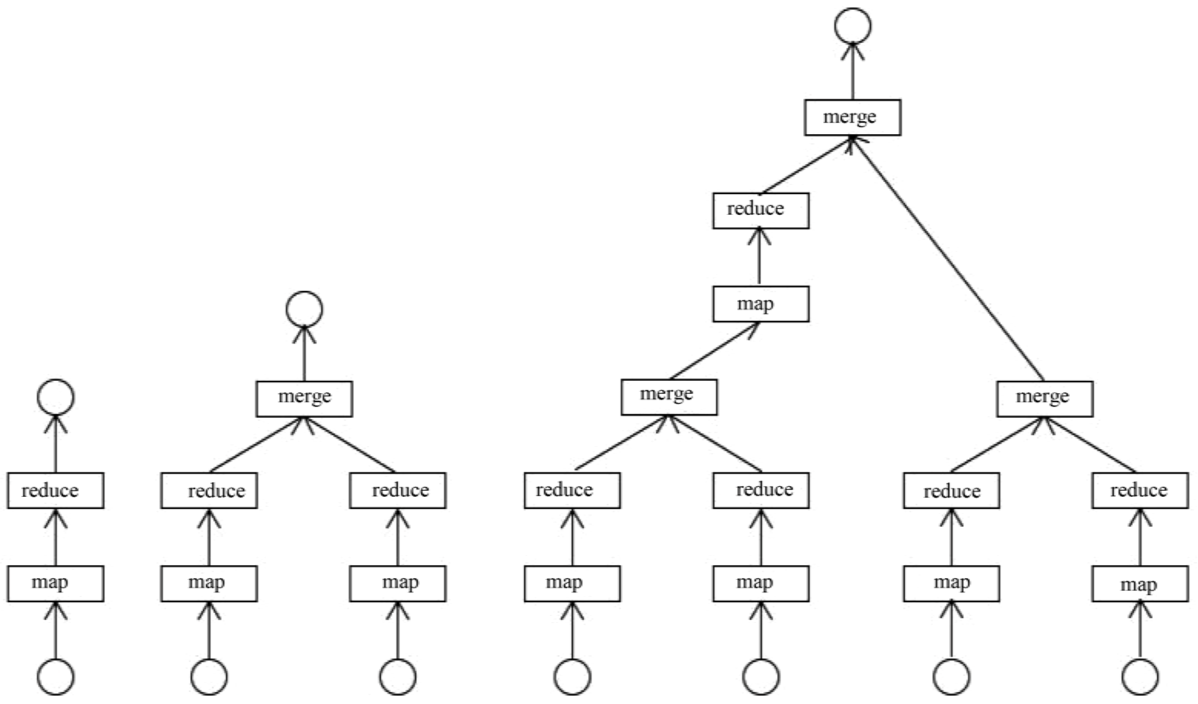


Figure 5. Workflow of Map-Reduce-Merge.

The module interactively moves data block in the node of high utilization to the point of the low utilization rate. In each iteration, the amount of move or receive of one point does not exceed the capacity threshold proportion. In this implementation, the node is classified as the high utilization, the average utilization and underutilized. According to the amount of use of each node, transfer the load among nodes to balance the cluster.

Load balancing module works as follows:

1) To obtain detailed information of the neighboring nodes. When the the DataNode load is added to the threshold level, will send a request to the NameNode and Namenode will get load level information in the neighboring nodes of specific the DataNode. Then NameNode compare the load information, and then send the details of the idlest adjacent nodes to a specific DataNode.

2) The DataNode start work. Each DataNode compare its load to its nearest node. If its load level is higher than its neighboring nodes, it will randomly select its neighbors and send the request to the destination node.

5. Conclusions

Firstly, this paper simply introduces the concept and programming model of MapReduce. The MapReduce has many important features, such as high-throughput, high-performance, fault tolerance and ease of manageability [17] [18]. Among them, the most important features is the parallel programming abstraction for two simple primitives, Map and Reduce, so developers can easily work in the real-world data processing converted into parallel programs.

However, MapReduce cannot directly support the handling of heterogeneous data sets, this is a fatal flaw for the non-standardization of Web data processing. It can be solved effectively by Adding Merge stage on the basis of the original model. And the new programming framework of the Map-Reduce-Merge inherited the the original MapReduce characteristics. Finally, for some of the features for Web data, do further optimization and improvement on the part of the Map-Reduce-Merge model scheduling policy, workflow, and load balancing to enhance the framework operating efficiency while also expanded its versatility.

References

[1] Dean, R. and Ghemawat, A. (2004) MapReduce: Implified Data Processing on Large Cluster. *SDI*, 137-149.

[2] Ghemawat, N., Gobioff, H. and Leung, S.T. (2003) The Google File System. *Proceedings of the SOSP*'03, Bolton Landing, 19-22 October 2003, 29-43.

[3] DOUG CUTTING (2005) Scalable Computing with MapReduce. OSCON.

[4] Borthankur, D. (2007) The Hadoop Distributed File System: Architecture and Design. Apache Software Foundation. 5-14.

[5] Daniel Abadi, M., DeWitt, D.J., *et al*. (2010) MapReduce and Parallel DBMSs: Friends or Foes. *Communications of the ACM*, **53**.

[6] Hadoop, T.W. (2009) The Definitive Guide. O'Reilly Media, 153-174.

[7] Zaharia, M., Konwinski, A. and Joseph, A.D. (2008) Improving MapReduce Performance in Heterogeneous Environment. *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, San Diego, 8-10 December 2008, 9-15.

[8] Becerra, Y., Beltran, V., Carrera, D., Gonzalez, M., Torres, J. and Ayguade, E. (2009) Speeding Up Distributed MapReduce Applications Using Hardware Accelerators. *Proceedings of the* 2009 *International Conference on Parallel Processing*, Vienna, 22-25 September 2009, 42-49. http://dx.doi.org/10.1109/ICPP.2009.59

[9] Fei, X., Lu, S. and Lin, C. (2009) A MapReduce-Enabled Scientific Workflow Composition Framework. *Proceedings of the IEEE International Conference on Web Services*, Los Angeles, 6-10 July 2009, 663-670.

[10] Hadoop 0.20 Documentation, Capacity Scheduler.

[11] Hadoop 0.20 Documentation, Fair Scheduler.

[12] Tian, C., Zhou, H., He, Y. and Zha, L. (2009) A Dynamic MapReduce Scheduler for Heterogeneous Workloads. *Proceedings of the* 8th *International Conference on Grid and Cooperative Computing*, Lanzhou, 27-29 August 2009, 218-224.

[13] Dean, J. and Ghemawat, S. (2004) MapReduce: Simplified Data Processing on Large Clusters. *Proceedings of OSDI*'04, San Francisco, 5 December 2004, 137-150.

[14] Pike, R., Dorward, S., Griesemer, R., *et al*. (2005) Interpreting the Data: Parallel Analysis with Sawzall. *Scientific Programming*, **13**, 227-298. http://dx.doi.org/10.1155/2005/962135

[15] Lammel, R. (2006) Google's MapReduce Programming Model—Revisited. Draft, 26 p.

[16] Tian, F. and Chen, K. (2011) Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds. *Proceedings of the* 2011 *IEEE International Conference on Cloud Computing* (*CLOUD*), Washington DC, 4-9 July 2011, 155-162.

[17] Kim, K., Jeon, K., Han, H., Kim, S., Jung, H., Yeom, H.Y. and Bench, M.R. (2008) A Benchmark for MapReduce Framework. Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems, Victoria, 8-10 December 2008, 11-18. http://dx.doi.org/10.1109/ICPADS.2008.70

[18] Kim, K., Jeon, K., Han, H., Kim, S., Jung, H. and Yeom, H.Y. (2008) Mrbench: A Benchmark for MapReduce Framework. *Proceedings of the* 2008 14*th IEEE International Conference on Parallel and Distributed Systems*, Melbourne, 8-10 December 2008, 11-18.