Time-Space Scheduling in the MapReduce Framework

Zhuo Tang, Lingang Jiang, Ling Qi, Kenli Li, and Keqin Li

College of Information Science and Engineering, Hunan University, Changsha 410082, China

1. Introduction

Data are representations of information, and the information content of data is generally believed to be valuable, and data form the basis of information systems. Using computers to process data, extracting information is a basic function of information systems. In today's highly information-oriented society, Web can be said to be currently the largest information system, of which the data are massive, diverse, heterogeneous, and dynamically changing. Using Hadoop to rapidly extract useful information from massive data of an enterprise has become an efficient method for programmers in the process of application development.

The significance of big data is to analyze people's behavior, intention, and preference in the growing and popular social networks. It is also to process data with non-traditional structures and to explore their meanings. Big data is often used to describe a company's large amount of unstructured and semi-structured data. Using analysis to create these data in a relational database for downloading will spend too much time and money. Big data analysis and cloud computing are often linked together, because real-time analysis of large data requires a framework similar to MapReduce to assign works for hundreds or even thousands of computers. After several years of criticism, questioning, discussion, speculation, big data finally ushered in the era belonging to it.

Hadoop presents MapReduce as an analytics engine, and under the hood it uses a distributed storage layer referred to as the *Hadoop distributed file system* (HDFS). As an open source implementation of MapReduce, Hadoop is so far one of the most successful realizations of large-scale data-intensive cloud computing platforms. It has been realized that when and where to start the reduce tasks are the key problems to enhance the MapReduce performance.

For *time scheduling* in MapReduce, the existing work may result in a block of reduce tasks. Especially, when the map tasks' output is large, the performance of a MapReduce task scheduling algorithm will be influenced seriously. Through analysis for the current MapReduce scheduling mechanism, Section 3 in this chapter illustrates the reasons of system slot resource wasting, which results in reduce tasks waiting around. Then, the section proposes a self-adaptive reduce task scheduling policy for reduce tasks' start times in the Hadoop platform. It can decide the start time point of each reduce task dynamically according to each job context, including the task completion time and the size of map output.

Meanwhile, another main performance bottleneck is caused by all-to-all communications between mappers and reducers, which may saturate the top-of-rack switch and inflate job execution time. The bandwidth between two nodes is dependent on their relative locations in the network topology. Thus, moving data repeatedly to remote nodes becomes the bottleneck. For this bottleneck, reducing cross-rack communication will improve job performance. Current researches prove that moving task is more efficient than moving data [1], especially in the Hadoop distributed environment, where data skews are widespread.

Data skew is an actual problem to be resolved for MapReduce. Existing Hadoop's reduce task scheduler is not only locality unaware, but also partitioning skew unaware. The parallel and distributed computation features may cause some unforeseen problems. Data skew is a typical such problem, and the high runtime complexity amplifies the skew and leads to highly varying execution times of the reducers. Partitioning skew causes shuffle skew, where some reduce tasks receive more data than others. The shuffle skew problem can degrade performance, because a job might get delayed by a reduce task fetching large input data. In the presence of data skew, we can use a reducer placement method to minimize all-to-all communications between mappers and reducers, whose basic idea is to place related map and reduce tasks on the same node or cluster or rack.

Section 4 of this chapter addresses *space scheduling* in MapReduce. We analyze the source of data skew and conclude that partitioning skew exists within certain Hadoop applications. The node at which a reduce task is scheduled can effectively mitigate the shuffle skew problem. In these cases, reducer placement can decrease the traffic between mappers and reducers and upgrade system performance. Some algorithms are released, which synthesize the network locations and sizes of reducers' partitions in their scheduling decisions in order to mitigate network traffic and improve MapReduce performance. Overall, Section 4 introduces several ways to avoid scheduling delay, scheduling skew, poor system utilization, and low degree of parallelism.

Some typical applications are discussed in this chapter. At present, biomedical literature has an enormous quantity and continues to increase at high speed. People urgently need some automatic tools to process and analyze the biomedical literature. In the current methods, the model training time increases sharply when dealing with large-scale training samples. How to increase the efficiency of named entity recognition in biomedical big data becomes one of the key problems in biomedical text mining. For the purposes of improving the recognition performance and reducing the training time, through implementing the model training process based on MapReduce, Section 5 of this chapter proposes an optimization method for two-phase recognition using conditional random fields (CRFs) with some new feature sets.

2. Overview of Big Data Processing Architecture

MapReduce is an excellent model for distributed computing, introduced by Google in 2004 [2]. It has emerged as an important and widely used programming model for distributed and parallel computing, due to its ease of use, generality, and scalability. Among its open source implementation versions, Hadoop has been widely used in industry around the whole world [3]. It has been applied to the production environments, such as Google, Yahoo, Amazon, Facebook, and so on. Because of the short development time, Hadoop can be improved in many aspects, such as the problems of intermediate data management and reduce task scheduling [4].

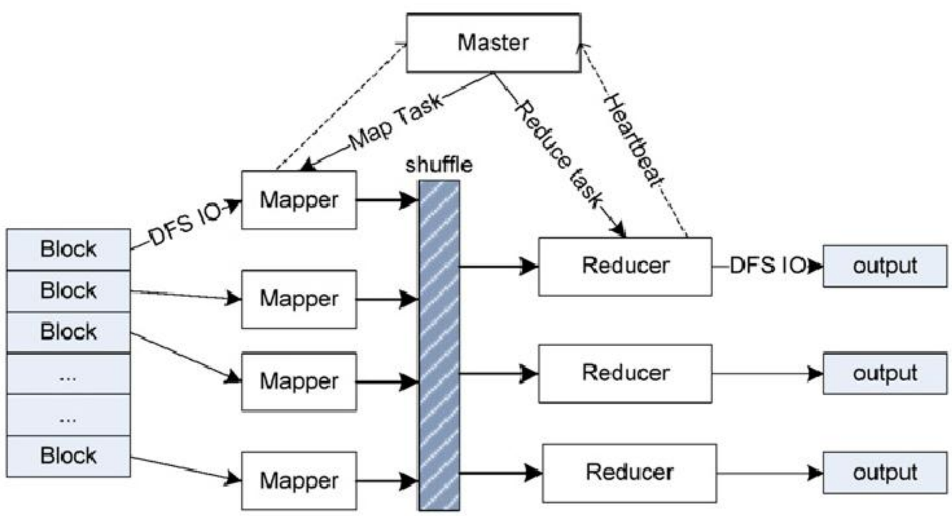


Figure 1. The typical process of the MapReduce.

As shown in Fig.1, map and reduce are two sections in a MapReduce scheduling algorithm. In Hadoop, each task contains three function phases, i.e., copy, sort, and reduce [5]. The goal of the copy phase is to read the map tasks' output. The sort phase is to sort the intermediate data, which are the output from map tasks and will be the input to the reduce phase. Finally, the eventual results are produced through the reduce phase, where the copy and sort phases are to preprocess the input data of the reduce phase. In real applications, copying and sorting may cost considerable amount of time, especially in the copy phase. In the theoretical model, the reduce functions start only if all map task are finished [6]. However, in the Hadoop implementation, all copy actions of reduce tasks will start when the first map action is finished [7]. But in slot duration, if there is any map task still running, the copy actions will wait around. This will lead to the waste of reduce slot resources.

In traditional MapReduce scheduling, reduce tasks should start when all the map tasks are completed. In this way, the output of map tasks should be read and written to the reduce tasks in the copy process [8]. However, through the analysis of the slot resource usage in the reduce process, this chapter illustrates that data transfer will result in slot idle and delay. In particular, when the map tasks' output becomes large, the performance of MapReduce scheduling algorithms will be influenced seriously [9]. When multiple tasks are running, inappropriate scheduling of the reduce tasks will lead to the situation where other jobs in the system cannot be scheduled timely. These are the stumbling blocks of Hadoop popularization.

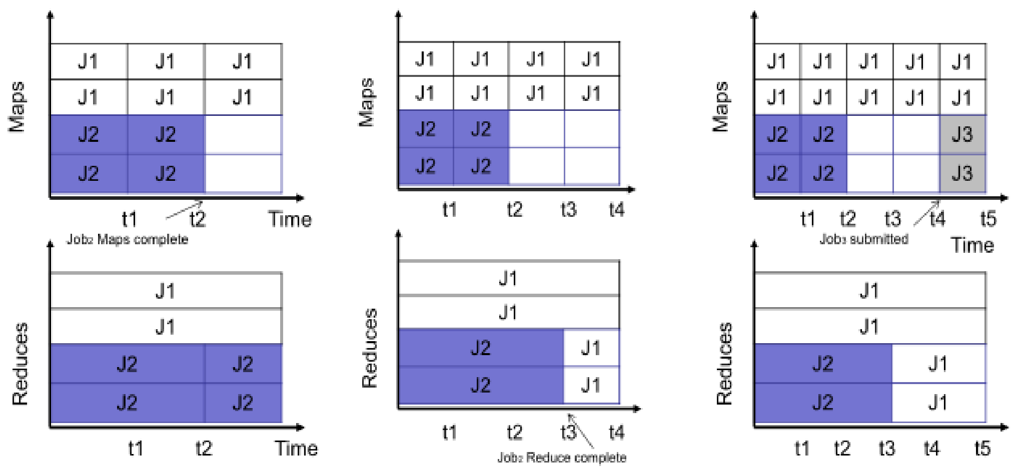
A user needs to serve two functions in the Hadoop framework, i.e., mapper and reducer, to process data. Mappers produce a set of files and send to all the reducers. Reducers will receive files from all the mappers, which is an all-to-all communication model. Hadoop runs in a datacenter environment in which machines are organized in racks. Cross-rack communication happens if a mapper and a reducer reside in different racks. Every cross-rack communication needs to travel through the root switch and hence the all-to-all communication model becomes a bottleneck.

This chapter points out the main affecting factors for the system performance in the MapReduce framework. The solutions to these problems constitute the content of the proposed time-space scheduling algorithms. In Section 3, we present a self-adaptive reduce task scheduling algorithm to resolve the problem of slot idle and waste. In Section 4, we analyze the source of data skew in MapReduce, and introduce some methods to minimize cross-rack communication and MapReduce traffic. To show the application of this advanced MapReduce framework, in Section 5, we describe a method to provide the parallelization of model training in named entity recognition in biomedical big data mining.

3. Self-Adaptive Reduce Task Scheduling

3.1 Problem Analysis

Through studying reduce task scheduling in the Hadoop platform, this chapter proposes an optimizing policy called *self-adaptive reduce scheduling* (SARS) [10]. This method can decrease the waiting around of copy actions and enhance the performance of the whole system. Through analyzing the details of the map and reduce two-phase scheduling process at the runtime of the MapReduce tasks[11], SARS can determine the start time point of each reduce task dynamically according to each job's context, such as the task completion time, the size of map output[12], etc. This section makes the following contributions: (1) the analysis for the current MapReduce scheduling mechanism and illustration of the reasons of system slot resource wasting which results in reduce tasks waiting around; (2) the development of a method to determine the start times of reduce tasks dynamically according to each job context, including the task completion time and the size of map output; (3) the description of an optimizing reduce scheduling algorithm which decreases the reduce completion time and system average response time in a Hadoop platform.



(a) Job2 map tasks finished (b) Job2 reduce tasks finished (c) Job3 submitted

Figure 2. The performance of the policies with respect to various graph sizes.

Hadoop allows the user to configure the job, submit it, control its execution, and query the state. Every job consists of independent tasks, and each task needs to have a system slot to run. Fig.2 shows the time delay and slot resources waste problem in reduce task scheduling. Through Fig.2(a), we can know that Job_1 and Job_2 are the current running jobs, and at the initial time, each job is allocated two map slots to run respective tasks. Since the execution time of each task is not the same, as shown in Fig.2(a), the Job_2 finishes its map tasks at time t2. Because the reduce tasks will begin once any map task finishes, from the duration t1 to t2, there are two reduce tasks from Job_1 and Job_2 which are running respectively. As indicated in Fig.2(b), at time t3, when all the reduce tasks of Job_2 are finished, two new reduce tasks from Job_1 are started. Now all the reduce slot resources are taken up by Job_1. As shown in Fig.2(c), at the moment t4, when Job_3 starts, two idle map slots can be assigned to it, and the reduce tasks from this job will then start. However, we can find that all reduce slots are already occupied by Job_1, and the reduce tasks from Job_3 have to wait for slot release.

The root cause of this problem is that reduce task of Job_3 must wait for all the reduce tasks of Job_1 to be completed, as Job_1 takes up all the reduce slots and Hadoop system does not support preemptive action acquiescently. In early algorithm design, a reduce task can be scheduled once any map tasks are finished [13]. One of the benefits is that the reduce tasks can copy the output of the map tasks as soon as possible. But reduce tasks will have to wait before all map tasks are finished, and the pending tasks will always occupy the slot resources, so that other jobs which finish the map tasks cannot start the reduce tasks. All in all, this will result in long waiting of reduce tasks, and greatly increase the delay of Hadoop jobs.

In practical applications, a shared cluster environment often has different jobs in running which are from multiple users at the same time. If the above similar situation appears among the different users at the same time, and the reduce slot resources are occupied for a long time, the submitted jobs from other users will not be pushed ahead until the slots are released. Such inefficiency will extend the average response time of a Hadoop system, lower the resource utilization rate, and affect the throughput of a Hadoop cluster.

3.2 Runtime Analysis of MapReduce Jobs

Through the above analysis, one method to optimize the MapReduce tasks is to select an adaptive time to schedule the reduce tasks. By this means, we can avoid the reduce tasks' waiting around and enhance the resource utilization rate. This section proposes a self-adaptive reduce task scheduling policy, which gives a method to estimate the start time of a task, instead of the traditional mechanism where reduce tasks are started once any map task is completed.

The reduce process can be divided into the following several phases. Firstly, the reduce task requests to read each map output data in the copy phase, which belong to this reduce function in the map out data blocks. Next, in the sort process, these intermediate data are output to an ordered data set by merging, which are divided into two types. One type are the data in memory. When the data are read from the various maps at the same time, the data should be merged as the same keys. The other is as like the circle buffer. Because the memory belonging to the reduce task is limited, the data in the buffer should be written to disks regularly in advance.

In this way, subsequent data need to be merged by the data which are written into the disks earlier, the so called external sorting. The external sorting needs to be executed several times if the number of map tasks are large in the practical works. The copy and sort are customarily called the shuffle phase. Finally, after finishing the copy and sort process, the subsequent functions start, and the reduce tasks can be scheduled to the compute nodes.

3.3 A Method of Reduce Task Start Time Scheduling

Because Hadoop employs the greedy strategy to schedule the reduce tasks, to schedule the reduce tasks fastest, as described above, some reduces tasks will always take up the system resources without actually performing operations in a long time. Reduce task start time is determined by this advanced algorithm SARS (Self-Adaptive Reduce Scheduling). In this method, the start times of the reduce tasks are delayed for a certain duration to lessen the utilization of system resources. The SARS algorithm schedules the reduce tasks at a special moment, when some map tasks are finished but not all. By this means, how to select an optimal time point to start the reduce scheduling is the key problem of the algorithm. Distinctly, the optimum point can minimize the system delay and maximize the resource utilization.

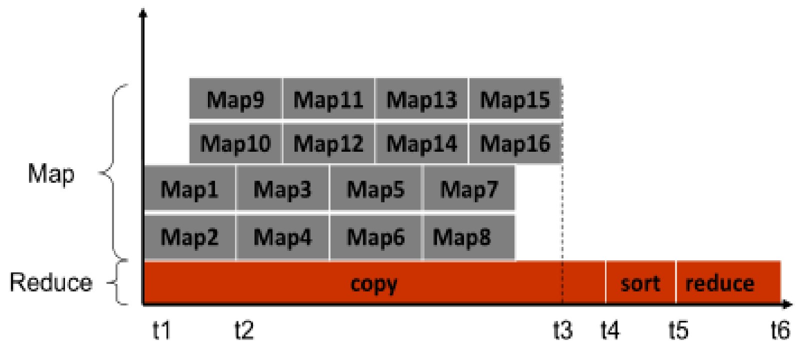


Figure 3. The default scheduling of reduce tasks.

As shown in Fig.3, assuming that Job_1 has 16 map tasks and one reduce task, and there are 4 map slots and only one reduce slot in this cluster system. Figures 3 and 4 describe the time constitution of the life cycle for a special job:

$$(FT_{lm}- STf_m) + (FT_{cp}- FT_{lm})+(FT_{lr}+ ST_{sr}). \qquad (3\text{-}1)$$

The denotations in Eq. (3-1) are defined as follows. FT_{lm} is the completion time of the last map task; STf_m is the start time of the first map task; FT_{cp} is the finish time of the copy phase; FT_{lr} is the finish time of reduce; ST_{sr} is the start time of reduce sort.

In Fig.3, t1 is the start time of Map1, Map2, and the reduce task. During t1 to t3, the main work of the reduce task is to copy the output from Map1 to Map14. The output of Map15 and Map16 will be copied by the reduce task from t3 to t4. The duration from t4 to t5 is so called the sort stage, which ranks the intermediate results according to the key values. The reduce function is called at the time t5, which continues from t5 to t6. Because during t1 to t3, in the copy phase, the reduce task only copies the output data intermittently, once any map task is completed, and for the most time it is always waiting around. We hope to make the copy operations completed at a concentrated duration, which can decrease the waiting time of the reduce tasks.

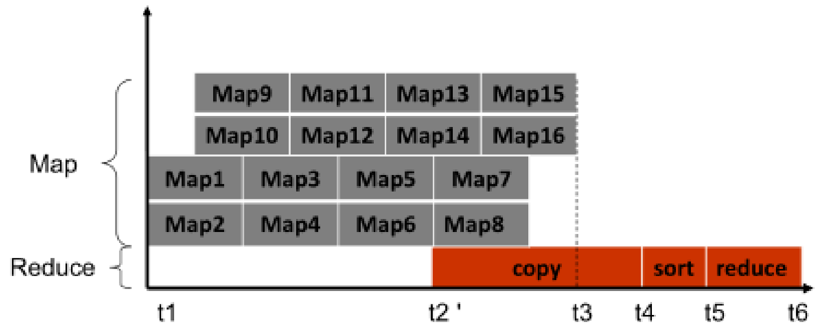


Figure 4. The scheduling method for reduce tasks in SARS.

As Fig.4 shows, if we can start the reduce tasks at t2’, which can be calculated using the following equations, and make sure these tasks can be finished before t6, then during t1 to t2’, the slots can be used by any other reduce tasks. But if we let the copy operation start at t3, because the output of all map tasks should be copied from t3, delay will be produced in this case. As shown in Fig.3, the copy phase starts at t2, which just collects the output of the map tasks intermittently. By contrast, the reduce task’s waiting time is decreased obviously in Fig.4, in which case the copy operations are started at t2’.

The SARS algorithm works by delaying the reduce processes. The reduce tasks are scheduled when part but not all of the map tasks are finished. For a special key value, if we assume that there are s map slots and m map tasks in the current system, and the completion time and the size of output data of each map task are denoted as t_map_i and m_out_j respectively, where $i, j \in [1,m]$. Then, we can know the amount of the map tasks data can be calculated as:

$$N\_m = \sum_{j=1}^{m} m\_out_j,\ j \in [1,m]. \tag{3-2}$$

In order to predict the time required to transmit the data, we define the speed of the data transmission from the map tasks to the reduce tasks as *transSpeed* in the cluster environment, and the number of concurrent copy threads with reduce tasks is denoted as *copyThread*. We denote the start time of the first map task and the first reduce task as $start_{map}$ and $start_{reduce}$ respectively. Therefore, the optimal start time of reduce tasks can be determined by following equation:

$$start_{reduce} = start_{map} + \frac{\sum_{i=1}^{m} t\_map_i}{s} - \frac{N\_m}{transSpeed \times copyThread}. \tag{3-3}$$

As shown by the time t2' in Fig.4, the most appropriate start time of a reduce task is when all the map tasks about the same key are finished, which is between the times when the first map is started and when the last map is finished. The second item in Eq. (3-3) denotes the required time of the map tasks, and the third item is the time for data transmission. Because the reduce tasks will be started before the copy processes, the time cost should be cut from the map tasks completion time. The waiting around of the reduce tasks may make the jobs in need of the slot resources not able to work normally. Through adjusting the reduce scheduling time, this method can decrease the time waste for data replication process and advance the utilization of the reduce slot resources effectively. Using the job's own characteristics to determine the reduce scheduling time can use the slot resources effectively. The improvement of these policies is especially important for the CPU-type jobs. For these jobs which need more CPU computing, the data I/O of the tasks are less, so more slot resource will be wasted in the default schedule algorithm.

4. Reduce Placement

As the mapper and reducer functions use an all-to-all communication model, this section presents some exiting and popular solutions in Sections 4.1-4.3, where we introduce several algorithms to optimize the communication traffic, which could increase the performance of data processing. In Sections 4.4-4.5, we mention the existence of data skew, and propose some methods based on space scheduling, i.e., reduce placement, to solve the problem of data skew.

4.1 Optimal Algorithms for Cross-Rack Communication Optimization

In Hadoop framework, a user needs to provide two functions, i.e., mapper and reducer, to process data. Mappers produce a set of files and send to all the reducers, and a reducer will receive files from all the mappers, which is an all-to-all communication model. Cross-rack communication [14] happens if a mapper and a reducer reside in different racks, which is very often in today's data center environments. Typically, Hadoop runs in a datacenter environment in which machines are organized in racks. Each rack has a top-of-rack switch and each top-of-rack switch is connected to a root switch. Every cross-rack communication needs to travel through the root switch and hence the root switch becomes a bottleneck [15]. MapReduce employs an all-to-all communication model between mappers and reducers. This results in saturation of network bandwidth of top-of-rack switch in the shuffle phase and straggles some reducers and increases job execution time.

There are two optimal algorithms to solve the reducer placement problem (RPP), and an analytical method to find the minimum (may not be feasible) solution of RPP, which considers the placement of reducers to minimize cross-rack traffic. One algorithm is a *greedy algorith*m [16], which assigns one reduce task to a rack at a time. When assigning a reduce task to a rack, it chooses the rack which incurs the minimum total traffic (up and down) if the reduce task is assigned to that rack. The second algorithm, called *binary se*arch [17], uses binary search to find the minimum bound of the traffic function for each rack, and then uses that minimum bound to find the number of reducers on each rack.

4.2 Locality-Aware Reduce Task Scheduling

MapReduce assumes the master-slave architecture and a tree-style network topology [18]. Nodes are spread over different racks encompassed in one or many data centers. A salient point is that the bandwidth between two nodes is dependent on their relative locations in the network topology. For example, nodes that are in the same rack have higher bandwidth between them as opposed to nodes that are off-rack. As such, it pays to minimize data shuffling across racks. The master in MapReduce is responsible for scheduling map tasks and reduce tasks on slave nodes after receiving requests from slaves for that regard. Hadoop attempts to schedule map tasks in proximity to input splits in order to avoid transferring them over the network. In contrast, Hadoop schedules reduce tasks at requesting slaves without any data locality consideration. As a result, unnecessary data might get shuffled in the network causing performance degradation.

Moving data repeatedly to distant nodes is becoming the bottleneck[19]. We rethink reduce task scheduling in Hadoop and suggest making Hadoop's reduce task scheduler aware of partitions' network locations and sizes in order to mitigate network traffic. There is a practical strategy that leverages network locations and sizes of partitions to exploit data locality, named *locality-aware reduce task scheduler* (LARTS) [18]. In particular, LARTS attempts to schedule reducers as close as possible to their maximum amount of input data and conservatively switches to a relaxation strategy seeking a balance between scheduling delay, scheduling skew, system utilization, and parallelism. LARTS attempts to collocate reduce tasks with the maximum required data computed after recognizing input data network locations and sizes. LARTS adopts a cooperative paradigm seeking good data locality while circumventing scheduling delay, scheduling skew, poor system utilization, and low degree of parallelism. We implemented LARTS in Hadoop-0.20.2. Evaluation results show that LARTS outperforms the native Hadoop reduce task scheduler by an average of 7%, and up to 11.6%.

4.3 MapReduce Network Traffic Reduction

Informed by the success and the increasing prevalence of MapReduce, we investigate the problems of data locality and partitioning skew present in the current Hadoop implementation and propose the *center-of-gravity reduce scheduler* (CoGRS) algorithm [20], a locality-aware and skew-aware reduce task scheduler for saving MapReduce network traffic. CoGRS attempts to schedule every reduce task *R* at its center-of-gravity node determined by the network locations of *R*'s feeding nodes and the skew in the sizes of *R*'s partitions. Notice that the center-of gravity node is computed after considering partitioning skew as well.

The network is typically a bottleneck in MapReduce-based systems. By scheduling reducers at their center-of-gravity nodes, we argue for reduced network traffic which can possibly allow more MapReduce jobs to co-exist in the same system. CoGRS controllably avoids scheduling skew, a situation where some nodes receive more reduce tasks than others, and promotes pseudo-asynchronous map and reduce phases. Evaluations show that CoGRS is superior to native Hadoop. When Hadoop schedules reduce tasks, it neither exploits data locality nor addresses partitioning skew present in some MapReduce applications. This might lead to increased cluster network traffic.

We implemented CoGRS in Hadoop-0.20.2 and tested it on a private cloud as well as on Amazon EC2. As compared to native Hadoop, our results show that CoGRS minimizes off-rack network traffic by average of 9.6% and 38.6% on our private cloud and on an Amazon EC2 cluster, respectively. This reflects on job execution times and provides an improvement of up to 23.8%.

Partitioning skew refers to the significant variance in intermediate keys' frequencies and their distribution across different data nodes. In essence, a reduce task scheduler can determine the pattern of the communication traffic in the network, affect the quantity of shuffled data, and influence the runtime of MapReduce jobs.

4.4 The Source of MapReduce Skews

Over the last few years, MapReduce has become popular for processing massive data sets. Most research in this area consider simple application scenarios like log file analysis, word count, and sorting, and current systems adopt a simple hashing approach to distribute the load to the reducers. However, processing massive amounts of data exhibit imperfections to which current MapReduce systems are not geared. The distribution of scientific data is typically skewed [21]. The high runtime complexity amplifies the skew and leads to highly varying execution times of the reducers.

There are three typical skews in MapReduce. (1) *Skewed key frequencies* – If some keys appear more frequently in the intermediate data tuples, the number of tuples per cluster owned will be different. Even if every reducer receives the same number of clusters, the overall number of tuples per reducer received will be different. (2) *Skewed tuple sizes* – In applications which hold complex objects within the tuples, unbalanced cluster sizes can arise from skewed tuple sizes. (3) *Skewed execution times* – If the execution time of the reducer is worse than linear, processing a single large cluster may take much longer than processing a higher number of small clusters. Even if the overall number of tuples per reducer is the same, the execution times of the reducers may differ.

According to those skew types, we propose several processes to improving the performance of MapReduce.

4.5 Reduce Placement in Hadoop

In Hadoop, map and reduce tasks typically consume large amount of data, and the total intermediate output (or total reduce input) size is sometimes equal to the total input size of all map tasks (e.g., sort) or even larger (e.g., 44.2% for K-means). For this reason, optimizing the placement of reduce tasks to save network traffic becomes very essential as optimizing the placement of map tasks, which is already well understood and implemented in Hadoop systems.

This section explores scheduling to ensure that the data that a reduce task handles the most are localized, so that it can save traffic cost and diminish data skew [22].

Sampling – Input data is loaded into a file or files in a distributed file system (DFS) where each file is partitioned into smaller chunks, called input splits. Each split is assigned to a map task. Map tasks process splits [23], and produce intermediate outputs which are usually partitioned or hashed to one or many reduce tasks. Before a MapReduce computation begins with a map phase, where each input split is processed in parallel, a random sample of the required size will be produced. The split of samples are submitted to the auditor group; meanwhile, the master and map tasks will wait for the results of the auditor.

Auditor Group – The auditor group (AG) carries out a statistical and predicted test to calculate the distribution of reduce tasks, and then start the reduce VM [24] at the appropriate place in the PM. The AG will receive several samples, and then will assign its members which contain map and reduce tasks to them. The distribution of intermediate key/value pairs which adopt a hashing approach to distribute the load to the reducers will be computed in reduces.

Placement of Reduce VM – The results of AG will decide the placement of reduce virtual machines (VM). For example, in Fig.5, if 80% key/value pairs of reduce 1 come from map 2 and the remaining intermediate results are from map 1, the VM of reduce 1 will be started in the physical machine (PM) which contains the VM of map 2. Similarly, the VM of reduce 2 will be started in the PM which includes the VM of map 1.

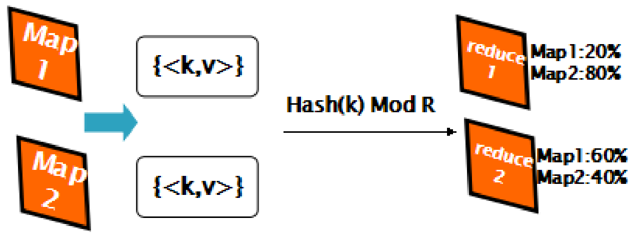


Figure 5. The intermediate results distribution in reduce tasks.

5. Named Entity Recognition in Biomedical Big Data Mining: A Case Study

Based on the above study of time-space Hadoop MapReduce scheduling algorithms, we present a case study in the field of biomedical big data mining. Compared to traditional methods and general MapReduce for data mining, our project makes originally inefficient algorithm become time-bearable in the case of integrating the above scheduling algorithms.

5.1 Biomedical Big Data

In the past several years, massive data have been accumulated and stored in different forms, whether in business enterprises, scientific research institutions, or government agencies. But when facing with more and more rapid expansion of the databases, people cannot set out to obtain and understand valuable knowledge within the big data.

The same situation has happened in the biomedical field. As one of the most concerned areas, especially after the human genome project (HGP), literature in biomedicine has appeared in large numbers, reaching an average of 600,000 or more per year [25]. Meanwhile, the completion of the human genome project has produced large human gene sequence data. In addition, with the fast development of science and technology in recent years, more and more large-scale biomedical experiment techniques, which can reveal the law of life activities on the molecular level, must use the big data from the entire genome or the entire proteome, which results in huge amount of biological data. These mass biological data contain a wealth of biological information, including significant gene expression situation and protein-protein interaction. What is more, a disease network, which contains hidden information associated with the disease and gives biomedical scientists the basis of hypothesis generation, is constructed based on disease relationship mining in these biomedical data.

However, the most basic requirements for biomedical big data processing are difficult to meet efficiently. For example, keyword searching in biomedical big data or the Internet can only find lots of relevant file lists, and the accuracy is not high, so that a lot of valuable information contained in the text cannot be directly shown to the people.

5.2 Biomedical Text Mining and Named Entity Recognition

In order to explore the information and knowledge in the biomedical big data, people integrate mathematics, computer science, and biology tools, which promote the rapid development of large-scale biomedical text mining. It refers to the biomedical big data analysis process of deriving high-quality information that is implicit, previously unknown, and potentially useful from massive biomedical data.

Current research emphasis on large-scale biomedical text mining is mainly composed of two aspects, i.e., information extraction and data mining. Specifically, it includes biomedical named entity recognition (Bio-NER), relation extraction, text classification, and integration framework of the above work.

Biomedical named entity recognition (Bio-NER) is the first and important and critical step in biomedical big data mining. It aims to help molecular biologists recognize and classify professional instances and terms, such as protein, DNA, RNA, cell_line, and cell_type. It is to locate and classify atomic elements with some special significance in biomedical text into predefined categories. The process of Bio-NER systems is structured as taking an unannotated block of text, and then producing an annotated block of text which highlights where the biomedical named entities are [26].

However, because of lots of unique properties in biomedical area, such as unstable quantity, non-unified naming rules, complex form, the existence of ambiguity and so on, Bio-NER is not mature enough, especially it takes much time. Most current Bio-NER systems are based on machine learning which need multiple iterative calculations from corpus data. Therefore, it is computationally intensive and seriously increases recognition time, including model training and inference. For example, it takes almost 5 hours for the CRFs model training process using Genia4ER training corpus which is only about 14MB [27]. How do we confront tens of thousands of biomedical text data volume? How do we cope with unbearable wait of recognition for a long long time? It is natural to seek for distributed computing and parallel computing to solve the problem.

5.3 MapReduce for Conditional Random Fields

Conditional random fields (CRFs) is an important milestone in the field of machine learning, put forward in 2001 by John Lafferty *et al*. [28]. CRFs, a kind of discriminant model and an undirected graph model at the same time, defines a single logarithmic linear distribution for a joint probability of entire label sequence based on a given particular observation sequence. The model is widely used in natural language processing (NLP), including *named entity recognition* (NER), part-of-speech tagging, and so on.

Figure 6 shows the CRFs model which computes the conditional probability $p(\vec{y}\,|\,\vec{x})$ of an output sequence $\vec{y}=(y_1,y_2,...,y_n)$ under the condition of a given input sequence $\vec{x}=(x_1,x_2,...,x_n)$.

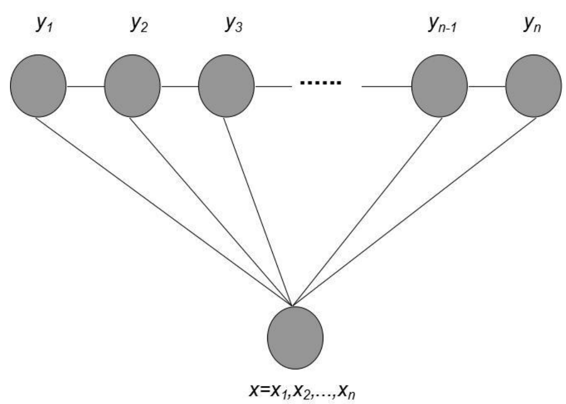


Figure 6. Liner CRFs.

Liner CRFs which is used in Bio-NER is as follows:

$$P(\vec{y}\,|\,\vec{x})=\frac{1}{Z(\vec{x})}.\exp(\sum_{i=1}^{n}\sum_{k=1}^{K}\lambda_k f_k(\vec{x},i,y_{i-1},y_i)), \tag{5-1}$$

where

$$Z(\vec{x})=\sum_{y}\exp(\sum_{i=1}^{n}\sum_{k=1}^{K}\lambda_k f_k(\vec{x},i,y_{i-1},y_i)), \tag{5-2}$$

and i is the position in the input sequence $\vec{x}=(x_1,x_2,...,x_n)$, and λ_k is a weight of a feature that does not depend on location i, and $\{f_k(\vec{x},i,y_{i-1},y_i)\}_{k=1}^{K}$ are feature functions.

For the training process of the CRFs model, it is to seek for the parameter $\vec{\lambda}=(\lambda_1,\lambda_2,...,\lambda_K)$ which is most in accordance with the training data $T=\{(\vec{x}_i,\vec{y}_i)\}_{i=1}^{N}$.

Presume every $(\vec{x},\vec{y})$ is independently and identically distributed. The parameter is obtained generally in this way:

$$L(\lambda)=\sum_{T}\log P(y\,|\,x)\,. \tag{5-3}$$

When the log-likelihood function $L(\lambda)$ reaches the maximum value, the parameter is almost the best. However, to find the parameter to maximize the training data likelihood, there is no closed-form solution. Hence, we adopt parameter estimation, i.e., the L-BFGS algorithm [29], to find the optimum solution.

To find the parameter $\vec{\lambda}=(\lambda_1,\lambda_2,...,\lambda_K)$ to make convex function $L(\lambda)$ reach the maximum, algorithm L-BFGS makes its gradient vector $\nabla L=(\frac{\partial L}{\partial \lambda_1},\frac{\partial L}{\partial \lambda_2},...,\frac{\partial L}{\partial \lambda_K})$ $\vec{0}$ by iterative computations with initial value $\lambda_0=0$ at first. Researches show that the first step, that is to calculate ∇L_i which is on behalf of the gradient vector in iteration i, calls for much time. Therefore, we focus on the optimized improvement for it.

Every component in ∇L_i is computed as follows:

$$\frac{\partial \grave{L}(\lambda)}{\partial \lambda_k}=\sum_{T}[\sum_{i=1}^{n} f_k(\vec{x},i,y_{i-1},y_i)-\sum_{\grave{y}}P(y'\,|\,\vec{x})\sum_{i=1}^{n} f_k(\vec{x},i,y'_{i-1},y'_i)]-\frac{\lambda_k}{\sigma^2}\,. \tag{5-4}$$

It can be linked with every ordered pair $(\vec{x},\vec{y})$ within $\sum_{T}$ which is mutually independent. So we can calculate the difference between $\sum_{i=1}^{n} f_k(\vec{x},i,y_{i-1},y_i)$ and $\sum_{\grave{y}}P(y'\,|\,\vec{x})\sum_{i=1}^{n} f_k(\vec{x},i,y'_{i-1},y'_i)$ on each of the input sequence in the training set *T*, and then put results of all the sequences together. As a result, it can be computed in parallel as shown in Fig.7.

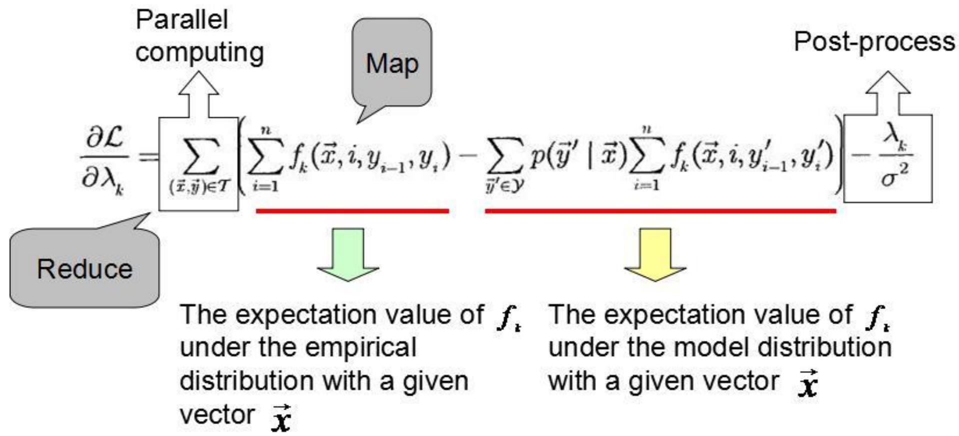


Figure 7. The MapReduce plan for computing component.

We split the calculation process in-house $\sum_T$ into several map tasks and summarize the results by a reduce task. And the difference between penalty term $\frac{\lambda_k}{\sigma^2}$ is designed to be the post-processing.

In the actual situation, it is impossible to schedule one map task for one ordered pair $(\vec{x}, \vec{y})$, because the number of ordered pairs in large-scale of training samples is too much and hard to estimate. We must syncopate the training data T into several small parts, and then start the MapReduce plan as shown in the above discussion.

For MapReduce Bio-NER application, the data skew leads to uneven load in the whole system. Any specific corpus has its own uneven distribution of the entity (as show in table below), resulting in the serious problem of data skew. And protean, artificial defined feature sets exacerbate the problem both in training and inference process.

Table 1. The proportion of each type of entities in the corpus JNLPBA2004

	Protein	DNA	RNA	Cell_line	Cell_type
Training Set	59.00%	18.58%	1.85%	7.47%	13.10%
Test Set	58.50%	12.19%	1.36%	5.77%	22.18%

Combined with schemes given in this chapter, it can be solved based on the modified Hadoop MapReduce. The implementation will further improve system performance on the MapReduce with time-space scheduling.

6. Concluding Remarks

As data are the basis of information systems, how to process data and extract information becomes one of the hottest topics in today's information society. This chapter introduces the MapReduce framework, an excellent distributed and parallel computing model. As its implementation, Hadoop plays a more and more important role in a lot of distributed application systems for massive data processing,

For the increasing data and cluster scales, to avoid scheduling delay, scheduling skew, poor system utilization, and low degree of parallelism, this chapter proposes some improved methods which focus on the time and space scheduling of reduce tasks in MapReduce.

Through analyzing the MapReduce scheduling mechanism, this chapter illustrates the reasons of system slot resource wasting which results in reduce tasks waiting around, and it proposes the development of a method detailing the start times of reduce tasks dynamically according to each job context, including the task completion time and the size of map output. There is no doubt that the use of this method will decrease the reduce completion time and system average response time in Hadoop platforms.

Current Hadoop schedulers often lack of data locality consideration. As a result, unnecessary data might get shuffled in the network causing performance degradation. This chapter addresses several optimizing algorithms to solve the problem of reduce placement. We make a Hadoop reduce task scheduler aware of partitions' network locations and sizes in order to mitigate network traffic and improve the performance of Hadoop.

Finally, a parallel biomedical data processing model using the MapReduce framework is presented as an application of the proposed methods. As USA proposed the human genome project (HGP), biomedical big data shows its unique position among the academics. A widely used CRFs model and an efficient Hadoop-based method, Bio-NER, have been introduced to explore the information and knowledge under the biomedical big data.

7. Bibliography

[1] J.Tan, S. Meng, X.Meng, L.Zhang. Improving reduce task data locality for sequential MapReduce jobs. International Conference on Computer Communications (INFOCOM), 2013 Proceedings IEEE, 14-19 April 2013, pp.1627-1635.

[2] J. Dean and S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of the ACM - 50th anniversary issue: 1958 - 2008 CACM Homepage archive Volume 51 Issue 1, January 2008, pp. 137 – 150.

[3] X. Gao, Q. Chen, Y. Chen, Q. Sun, Y. Liu, M. Li, A dispatching-rule-based task scheduling policy for MapReduce with multi-type jobs in heterogeneous environments, 2012 7th ChinaGrid Annual Conference (ChinaGrid), pp.17–24.

[4] J. Xie, F. Meng, H. Wang, H. Pan, J. Cheng, X. Qin, Research on scheduling scheme for Hadoop clusters, Procedia Computer Science, 2013, Volume: 18, pp. 2468–2471.

[5] Z. Tang, M. Liu, K.Q. Li , Y. Xu . A MapReduce-enabled scientific workflow framework with optimization scheduling algorithm, 2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), pp. 599–604.

[6] F. Ahmad, S. Lee, M. Thottethodi, T. N. Vijaykumar, MapReduce with communication overlap (MaRCO), Journal of Parallel and Distributed Computing, May 2013, Volume: 73, Issue: 5, pp. 608–620.

[7] M. Lin, L. Zhang, A. Wierman, J. Tan, Joint optimization of overlapping phases in MapReduce, Performance Evaluation, Volume: 70, Issue: 10, October 2013, pp. 720–735.

[8] Y. Luo, B. Plale, Hierarchical MapReduce programming model and scheduling algorithms, Cluster Computing and the Grid, IEEE International Symposium on, 13-16 May 2012, pp. 769–774.

[9] H. Mohamed, S. Marchand-Maillet, MRO-MPI: MapReduce overlapping using MPI and an optimized data exchange policy, Parallel Computing, In Press, Volume 39 Issue 12,10 September 2013, pp.851-866 .

[10] Z. Tang, L.G. Jiang, J. Q. Zhou, K.L. Li, K.Q. Li. A Self-Adaptive Scheduling Algorithm for Reduce Start Time. Future Generation Computer Systems. Manuscript under revision.

[11] Y. Luo, B. Plale, Hierarchical MapReduce programming model and scheduling algorithms, 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 13-16 May 2012, pp. 769–774.

[12] D. Linderman, D. Collins, H. Wang, H. Meng Merge: a programming model for heterogeneous multi-core systems, ASPLOS XIII Proceedings of the 13th international conference on Architectural support for programming languages and operating systems Volume 36 Issue 1, March 2008, pp: 287-296.

[13] B. Palanisamy,A. Singh,L. Liu,BLangston. Cura: A cost-optimized model for MapReduce in a cloud, IEEE International Symposium on Parallel and Distributed Processing (IPDPS), IEEE Computer Society, 20-24 May 2013, pp. 1275–1286.

[14] L. Ho,J. Wu, P. Liu. Optimal algorithms for cross-rack communication optimization in MapReduce framework. , 2011 IEEE International Conference on Cloud Computing (CLOUD), 4-9 July 2011, pp. 420-427.

[15] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar,and A. Goldberg. Quincy: fair scheduling for distributed computing clusters. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP), 11-14 October 2009, pp. 261-276.

[16] Wikipedia [EB/OL] http://en.wikipedia.org/wiki/Greedy_algorithm. 2013-09-14.

[17] Daniel Dominic Sleator,Robert Endre Tarjan. Self-adjusting binary search trees. Journal of the

ACM (JACM) JACM Homepage archive Volume 32 Issue 3, July 1985, pp. 652-686.

[18] M. Hammoud and M. F. Sakr. Locality-aware reduce task scheduling for MapReduce. Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 29 Nov – 1.Dec 2011, pp. 570-576.

[19] S. Huang, J. Huang, J. Dai, T. Xie, B. Huang. The HiBench benchmark suite: characterization of the MapReduce-based data analysis, Date Engineering Workshops (ICDEW), IEEE 26th International Conference on, 1-6 March 2010 , pp. 41-51.

[20] M. Hammoud, M. S Rehman, and M. F. Sakr. Center-of-gravity reduce task scheduling to lower MapReduce network traffic. Cloud Computing (CLOUD), IEEE 5th International Conference on, 24-29 June,2012, pp. 49-58.

[21] Y. C. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skew-resistant parallel processing of feature-extracting scientific user-defined functions, Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC), June 2010, pp. 75-86.

[22] P. Dhawalia, S.Kailasam; D.Janakiram. Chisel: A resource savvy approach for handling skew in MapReduce applications. Cloud Computing (CLOUD), IEEE Sixth International Conference on, 28 June -3 July 2013, pp. 652-660.

[23] R. Grover, M. J.Carey. Extending Map-Reduce for efficient predicate-based sampling. Data Engineering (ICDE), 2012 IEEE 28th International Conference on, pp: 486-497.

[24] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi. Evaluating MapReduce on virtual machines: the Hadoop case cloud computing. Lecture Notes in Computer Science, Volume 5931, 2009, pp: 519-528.

[25] Wikipedia, MEDLINE [EB/OL]. http://en.wikipedia.org/wiki/MEDLINE. 2013-09-14.

[26] J. Kim, T. Ohta, Y. Tsuruoka, Y. Tateisi, N. Collier. Introduction to the bio-entity recognition task at JNLPBA. Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (JNLPBA), August 2004, pp: 70-75.

[27] L. Li, R. Zhou, D. Huang. Two-phase biomedical named entity recognition using CRFs. Computational Biology and Chemistry, 2009, Volume 33 Issue 4, pp. 334–338.

[28] J. Lafferty, A. McCallum, F. Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In Proceedings of the 18th International Conference on Machine Learning (ICML), June 28 - July 1 2001, pp. 282-289.

[29] D. Liu, J Nocedal. On the limited memory BFGS method for large scale optimization. In: Mathematical Programming, Volume 45, Issue 1-3, August 1989, pp. 503-528.