Available online www.jocpr.com

Journal of Chemical and Pharmaceutical Research, 2014, 6(6):1270-1276

Research Article

ISSN : 0975-7384
CODEN(USA) : JCPRC5

Scientific data mining and processing using MapReduce in cloud environments

Kong Xiangsheng

Department of Computer & Information, Xinxiang University, Xinxiang, China

---

ABSTRACT

High-performance processing scientific data has enabled the development of digital resources and web-based services that facilitate uses of data beyond those that may have been envisioned by the original data producers. Scientific data processing systems must handle scientific data coming from real-time, high-throughput applications. Timely processing of scientific data is important and requires sufficient available resources to achieve high throughput and deliver accurate output results. Cloud Computing provides a low-priced way for small and medium sized enterprises to process scientific data. Based on Hadoop MapReduce of cloud computing, we propose the detailed procedure of scientific data processing algorithm which can improve the overall performance under the shared environment while retaining compatibility with the native Hadoop MapReduce in this paper.

Keywords: MapReduce; Cloud Computing; Hadoop; Distributed File System

---

INTRODUCTION

In many fields, such as astronomy, high-energy physics and bioinformatics, scientists need to analyze terabytes of data either from existing data resources or collected from physical devices [1]. Scientific applications are usually complex and data intensive. The current scientific computing landscape is vastly populated by the growing set of data-intensive computations that require enormous amounts of computational as well as storage resources and novel distributed computing frameworks. On the one hand, scientific data centers, libraries, government agencies, and other groups have moved rapidly to online digital data access and services, drastically reducing usage of traditional offline access methods. On the other hand, practices for storage and preservation of these digital data resources are far from the maturity and reliability achieved for traditional non-digital media [2]. On the industry front, Google Scholar and its competitors (e.g. Microsoft, CNKI, Baidu Library) have constructed large scale scientific data centers to provide stable web search services with high quality of response time and availability.

With the evolution in data storage, large databases have stimulated researchers from many areas especially machine learning and statistics to adopt and develop new techniques for data analysis. This has led to a new area of data mining and knowledge discovery. Applications of data mining in scientific applications have been studied in many areas. The focus of data mining in this area was to analyze data to help understanding the nature of scientific datasets. Automation of the whole scientific discovery process has not been the focus of data mining research.

The high demanding requirements on scientific data centers are also reflected by the increasing popularity of cloud computing [3]. Cloud computing is immensely appealing to the scientific community, who increasingly see it as being part of the solution to cope with burgeoning data volumes. With cloud, IT-related capabilities enable economies-of-scale in facility design and hardware construction [4]. There are several vendors that offer cloud computing platforms, representative systems for cloud computing include Google App Engine, Amazon Elastic Compute Cloud (EC2), AT&T's Synaptic Hosting, Rackspace, GoGrid and AppNexus. For large scale scientific data centers, cloud computing model is quite attractive because it is up to the cloud providers to maintain the hardware infrastructure.

Although the popularity of data centers is increasing, it is still a challenge to provide a proper computing paradigm which is able to support convenient access to the large scale scientific data for performing computations while hiding all low level details of physical environments. Within all the candidates, MapReduce is a broadly-useful computing paradigm that has recently gained prominence as robust, parallel implementations such as Google's MapReduce and others have been widely used to handle data-intensive operations across clusters in data centers. The MapReduce framework was originally proposed by Google in 2004 to deal with large scale web datasets and has been proved to be an effective computing paradigm for developing data mining, machine learning and search applications in data centers.

Hadoop was in production use at established and emerging web companies in 2006, it is an open source project and operates under the auspices of the Apache Software Foundation today. It was based on the Google MapReduce and is an open source implementation of the Google's MapReduce parallel processing framework [5]. With the Apache open source framework Hadoop the usage of MapReduce has been spread to a large number of other applications as well. I'm proposing in this paper that Cloud Computing and Hadoop MapReduce are better approaches and solutions for scientific data processing.

RELATED WORK

Both Cloud Computing and Hadoop MapReduce are two technologies that have gained a lot of popularity mainly due to its ease-of-use and its ability to scale up on demand. As a result, MapReduce scientific data processing is a popular application on the Cloud [6].

The MapReduce

Over the past five years MapReduce has attained considerable interest from both the database and systems research community. MapReduce is a programming model for data processing. The model is simple, but at the same time powerful enough to implement a great variety of applications.

MapReduce simplified the implementation of many data parallel applications by removing the burden from programmer such as tasks scheduling, fault tolerance, messaging, and data Processing [7]. Massively parallel programming frameworks such as MapReduce are increasingly popular for simplifying data processing on hundreds and thousands of cores, offering fault tolerance, linear scale-up, and a high-level programming interface. The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. With the MapReduce programming model, programmers only need to specify two functions: Map and Reduce [8]. MapReduce functions are as follows.

Map:(in_key,in_value)→{key_j, $value_j$ | j=1…k}
Reduce:(key, [$value_1$,…, $value_m$])→(key, final_value)

The input parameters of Map are in_key and in_value. The output of Map is a set of <key,value>. The input parameters of Reduce is (key, [$value_1$, ..., $value_m$]). After receiving the parameters, Reduce is run to merge the data which were get from Map and output (key, final_value).

The map function, written by the user, takes an input pair and produces a set of intermediate key/value pairs. It is an initial transformation step, in which individual input records can be processed in parallel [9]. The MapReduce library groups together all intermediate values associated with the same intermediate key i and passes them to the Reduce function.

The Reduce function, also written by the user, adds up all the values and produces a count for a particular key. It is an aggregation or summarization step, in which all associated records must be processed together by a single entity. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation.

There are five main roles: the engine, the master, the scheduling algorithm, mappers, and reducers [10]. Figure 1 shows a high level view of our architecture and how it processes the data.

The MapReduce engine is responsible for splitting the data by training examples (rows). The engine then caches the split data for the subsequent map-reduce invocations. The MapReduce engine is implemented as a plug-in component for the native Hadoop system and is compatible with both dedicated and shared environments.

Every scheduling algorithm has its own engine instance, and every MapReduce task will be delegated to its engine. Zaharia have proposed a new scheduling algorithm called LATE (Longest Approximation Time to End). They

showed that the Hadoop's current scheduler can cause severe performance degradation in heterogeneous environments such as virtualized data centers where uncontrollable variations in performance exist [11].

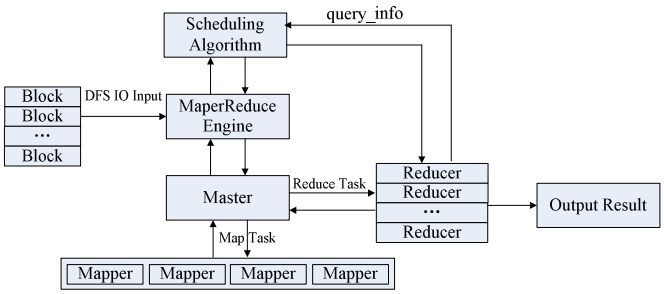

Figure 1. MapReduce Working Flow

The engine will run a master which acts as the coordinator responsible for the mappers and the reducers. The master who notifies the reducers to prepare to receive the intermediate results as their input is responsible for assigning the split data to different mappers, and collects the processed intermediate data from the mappers, and then in turn invokes the reducer to process it and return final results. Each mapper will process the data by parsing the key/value pair and then generate the intermediate result that is stored in its local file system. Reducers then use Remote Procedure Call (RPC) to read data from mappers.

The Hadoop

Hadoop stores the intermediate results of the computations in local disks, where the computation tasks are run, and then inform the appropriate workers to retrieve (pull) them for further processing. It hides the details of parallel processing and allows developers to write parallel processing programs that focus on their computation problem, rather than parallelization issues.

Hadoop relies on its own distributed file system called HDFS (Hadoop Distributed File System): a flat-structure distributed file system that store large amount of data with high throughput access to data on clusters. HDFS is a mimic of GFS (Google File System). Like GFS, HDFS has master/slave architecture, and multiple replicas of data are stored on multiple compute nodes to provide reliable and rapid computations [12]. As shown in figure 2 HDFS consists of a single NameNode and multiple DataNodes in a cluster.

A client accesses the file system on behalf of the user by communicating with the NameNode and DataNodes. The client presents a POSIX6-like file system interface, so the user code does not need to know about the details of the NameNode and DataNodes to work correctly.

DataNodes perform the dirty work on the file system. They store and retrieve blocks when they are told to (by clients or the NameNode) and they report back to the NameNode periodically with lists of blocks that they are storing. Without the NameNode, the file system cannot be used. In fact, if the machine running the NameNode goes down, all the files on the file system would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the DataNodes. For this reason, it is important to make the NameNode resilient to failure.

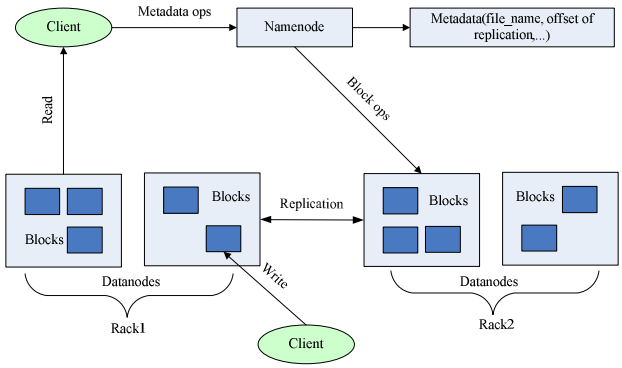

Figure 2. HDFS Architecture

NameNode is the master node that manages the file system name space which records the creation, deletion and modification of files by the users and regulates clients' access to files, while DataNodes manage storage directly

attached to each DataNode. DataNodes which are slave nodes perform block creation, deletion and replication of data blocks.

DATA MINING AND PROCESSING USING MAPREDUCE IN CLOUD ENVIRONMENTS

At the system's core, MapReduce is a style of parallel programming supported by capacity-on-demand clouds. A good illustrating example of how something like MapReduce works is to compute an inverted index in parallel for a large collection of Web pages stored in a cloud. Our system puts all essential functionality inside a cloud, while leaving only a simple Client at the experiment side for user interaction. In the cloud, we use Hadoop HDFS to store the scientific data.

Scientific Data Mining

Figure 3 outlines one way in which these tasks might be incorporated into an end-to-end data mining system for analyzing data from a science or engineering application. Starting with the raw data in the form of images or meshes, we successively process these data into more refined forms, enabling further processing of the data and the extraction of relevant information. The terms, such as Raw data and Target data where they are used to describe the process of Knowledge Discovery in Databases. In adapting this process to scientific data sets, we have retained the basic framework, but changed the tasks necessary to transform the data from one form to the next.

The raw data which are provided for data mining often need extensive processing before they can be input to a pattern recognition algorithm. These algorithms typically require, as input, the object in the data set, with each object described by a set of features. Thus, we first need to identify the objects in the data and extract features representing each object. In scientific data sets, the data may need to be processed before we can even identify the objects in the data. Once the data have been reduced through sampling and/or multiresolution techniques, complementary data sources have been fused, and the data enhanced to make it easier to extract the objects of interest, potential next steps in the scientific data mining process. Once the initial steps of preprocessing have been applied to the raw data, the objects in the data identified, and features representing them extracted, we have a matrix where the rows represent the data items or objects and the columns represent the features. We could also have considered the rows to be the features and the columns to be the objects.

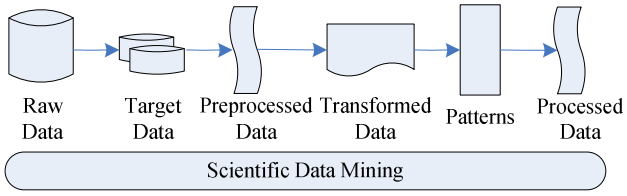

Figure 3. scientific data mining process

Parallel Computing over Clouds

MapReduce is a style of parallel programming supported by capacity-on-demand clouds [13]. A good illustrating example of how something like MapReduce works is to compute an inverted index in parallel for a large collection of Web pages stored in a cloud. Let's assume that each node i in the cloud stores Web pages $p_{i,1}$, $p_{i,2}$, $p_{i,3}$, …, and that a Web page p_j contains words (terms) $w_{j,1}$, $w_{j,2}$, $w_{j,3}$, … . A basic but important structure in information retrieval is an inverted index, that is, a list

(w_1; $p_{1,1}$, $p_{1,2}$, $p_{1,3}$, …)
(w_2; $p_{2,1}$, $p_{2,2}$, $p_{2,3}$, …)
(w_3; $p_{3,1}$, $p_{3,2}$, $p_{3,3}$, …),

where the list is sorted by the word w_j, and associated with each word w_j is a list of all Web pages p_i containing that word.

```
#include "mapreduce/mapreduce.h"
// User's map function
class WordCounter : public Mapper {
public:
virtual void Map(const MapInput& input) {
        const string& text = input.value();
        const int n = text.size();
        for (int i = 0; i < n; ) {
                // Skip past leading whitespace
                while ((i < n) && isspace(text[i]))
                        i++;
                // Find word end
                int start = i;
                while ((i < n) && !isspace(text[i]))
                        i++;
                        if (start < i)
                                Emit(text.substr(start,i-start),"1");
                        }
                }
};
REGISTER_MAPPER(WordCounter);
// User's reduce function
class Adder : public Reducer {
        virtual void Reduce(ReduceInput* input) {
        // Iterate over all entries with the same key and add the values
                int value = 0;
                while (!input->done()) {
                        value += StringToInt(input->value());
                        input->NextValue();
                }
                // Emit sum for input->key()
                Emit(IntToString(value));
        }
};
REGISTER_REDUCER(Adder);
int main(int argc, char** argv) {
        ParseCommandLineFlags(argc, argv);
        MapReduceSpecification spec;
        // Store list of input files into "spec"
        for (int i = 1; i < argc; i++) {
                MapReduceInput* input = spec.add_input();
                input->set_format("text");
                input->set_filepattern(argv[i]);
                input->set_mapper_class("WordCounter");
        }
        // Specify the output files:
        MapReduceOutput* out = spec.output();
        out->set_filebase("/gfs/test/freq");
        out->set_num_tasks(100);
        out->set_format("text");
        out->set_reducer_class("Adder");
        // Optional: do partial sums within map
        // tasks to save network bandwidth
        out->set_combiner_class("Adder");
        // Tuning parameters: use at most 2000
        // machines and 100 MB of memory per task
        spec.set_machines(2000);
        spec.set_map_megabytes(100);
        spec.set_reduce_megabytes(100);
        // run it
        MapReduceResult result;
        if (!MapReduce(spec, &result)) abort();
        // Done: 'result' structure contains info about
        //counters, time taken, number of machines used, etc.
        return 0;
      }
```

MapReduce uses a programming model that processes a list of <key, value> pairs to produce another list of <key', value'> pairs. The initial list of <key, value> pairs is distributed over the nodes in the cloud. In the map phase, each Web page pi is processed independently on its local node to produce an output list of multiple key-value pairs <w_j, p_i>, one for word w_j on the page.

Scientific Data Processing Algorithm on MapReduce

The scientific data consist of a value from an input array and its index in the input array. The master starts with sending such a message to each of the slaves [14]. Then the master waits for any slave to return a result. As soon as the master receives a result, it will insert the result into the output array and provide further work to the slave if any is available. As soon as all work has been submitted to the slaves, the master will just wait for the slaves to return their last result. In order to count the number of occurrences of each unique scientific word in a set of input files specified on the command line, we use the below program to process the scientific data.

PERFORMANCE ANALYSIS

In fact, the use of Hadoop allows to speed up calculations by a factor that equals the number of worker nodes, except for startup effects, which are relatively small when the execution time of individual tasks is large enough.

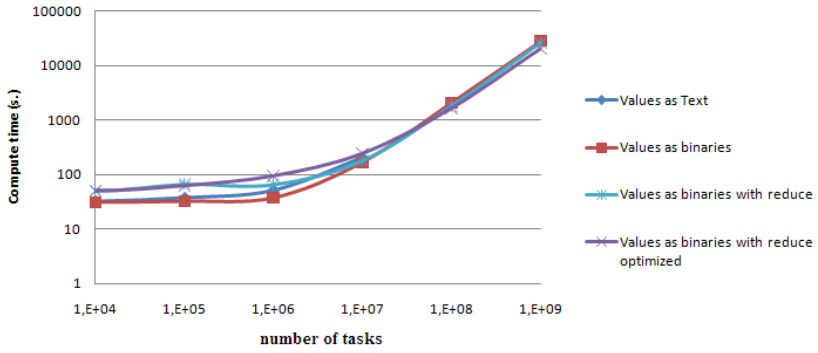

Figure 4. Test for scheduling overhead

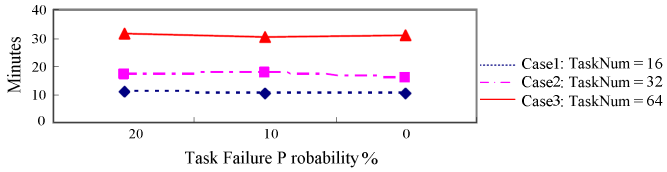

Figure 5. Test for the degree of fault tolerance

Figure 4 shows a test in which we run an increasing number of tasks with one folder per task. It can be seen that the total time increases, which means that there is an overhead for scheduling. Figure 5 shows how the system is resilient against failures. We again consider jobs with one folder per task. Each task has a failure rate of 0%, 10% or 20%. We artificially stop some of the tasks right after they start, with the probabilities specified. The figure shows that Hadoop is able to recover from these failures as expected. (It restarts tasks that have failed automatically.) The total time does not increase markedly when tasks fail, due to the fact that we let tasks fail immediately after they start. This shows that there is not much overhead associated with the fault tolerance mechanism. The figure also shows that 64 tasks take about twice the time of 32 tasks, as expected. Interestingly, 32 tasks take less than double the time of 16 tasks (on 20 cloud nodes), which is likely due to an overhead cost associated with starting and completing a job.

CONCLUSION

As cloud computing is such a fast growing market, differentcloud service providers will appear. It is clear to us that the traditional super computing centers consisting only of petascale computing resources are not sufficient to tackle the broad range of e-Science challenges. The cloud computing model, based on scientific data centers that scale well enough to support extremely large ondemand loads, are needed to

• Support large numbers of science gateways and their users
• Provide a platform that can support the creation of collaboration and data & application sharing spaces that can be used by virtual organizations
• Manage the computations that are driven by streams of scientific instrument data.

A reliable, geographically distributed data center equipped with a collection of software tools including cloud computing, parallel data analysis frameworks Hadoop MapReduce programming tools is needed.

REFERENCES

[1] Xiao Liu, The First CS3 PHD Symposium, **2010**, 49-51.
[2] Robert R. Downs; Robert S. Chen, *Journal of Digital Information*, **2010**(11), 35-42.
[3] Chao Jin; Rajkumar Buyya, Technical report, The University of Melbourne, Australia, **2008**, 1-33.
[4] Sangmi Lee Pallickara; Shrideep Pallickara; Marlon Pierce, Handbook of Cloud Computing Springer Science Business Media, **2010**, 517-527.
[5] R. Campbell; I. Gupta; M. Heath; S. Ko; M. Kozuch; M. Kunze; T. Kwan; K. Lai; H. Lee; M. Lyons, USENIX Workshop on Hot Topics in Cloud Computing, **2009**, 1-13.
[6] H. Yang; A. Dasdan; R. Hsiao; S. Parker, In Proceedings of the ACM SIGMOD International Conference on Management of Data, **2007**, 1029–1040.
[7] Bill Howe; Peter Lawson; Renee Bellinger; Erik W. Anderson; Emanuele Santos; Juliana Freire; Carlos Eduardo Scheidegger; Antonio Baptista; Claudio T. Silva, In Proceedings of the 4th IEEE International Conference on eScience, **2008**, 127–134.
[8] Zhifeng Xiao; Yang Xiao, The First International Workshop on Security in Computers, Networking and Communications, **2011**, 1099-1104.
[9] Jaliya Ekanayake; Shrideep Pallickara, Fourth IEEE International Conference on eScience, **2008**, 277-284.
[10] Cheng T. Chu; Sang K. Kim; Yi A. Lin; Yuanyuan Yu; Gary R. Bradski; Andrew Y. Ng; Kunle Olukotun, In *Advances in Neural Information Processing Systems* 19, **2006**, 281–288.
[11] Sangwon Seo; Ingook Jang1; Kyungchang Woo; Inkyo Kim; Jin-Soo Kim; Seungryoul Maeng, In Proceedings of the 2009 IEEE Cluster, **2009**, 1−8.
[12] B.Thirumala Rao; L.S.S.Reddy, *International Journal of Computer Applications*, **2011**(34), 28-32.
[13] DING Jian-li; YANG Bo, *International Journal of Digital Content Technology and its Applications*. **2011**(5), 236-243.
[14] W. Kleiminger; E. Kalyvianaki; P. Pietzuch, In Proceedings of the 6th International Workshop on Self Managing Database Systems, **2011**, 16-21.