Journal of Theoretical and Applied Information Technology
28th February 2013. Vol. 48 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HADOOP MAPREDUCE IN CLOUD ENVIRONMENTS FOR SCIENTIFIC DATA PROCESSING

[1]KONG XIANGSHENG
[1]Department of Computer & Information, Xin Xiang University, Xin Xiang, China
E-mail: fallsoft@163.com

ABSTRACT

For the problem of massive scientific data storage on the network, and based on the Hadoop and virtual technologies, the massive scientific data storage model based on cloud computing is proposed. The massive scientific data is applied to the Hadoop platform and processed by MapReduce, the key algorithm of cloud computing, and finally the data is stored in the virtual pool. This paper introduces a large scale scientific data processing method based on cloud computing, builds a dynamic, scalable, cost-effective, easy to use and high performance computing platform on a large of centralized or distributed inexpensive computer cluster, and creates a cloud computing based framework for large scale data processing model. 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, scientific data flow processing, Cloud Computing; Hadoop; Distributed File System*

1. INTRODUCTION

Today, scientific workflow produces huge amounts of scientific data which are stored in large data warehouses in digital form [1]. Scientific applications are usually complex and data intensive. In many fields, such as astronomy, high-energy physics and bioinformatics, scientists need to analyse terabytes of data either from existing data resources or collected from physical devices. The scientific analyses are usually computation intensive, hence taking a long time for execution.

In addition to enabling the creation and analysis of scientific data, the digital form facilitates much greater data sharing and reusing by others, which has led to the spread of data-driven science practices within scientific communities. Along with these new opportunities for sharing and using scientific data come new challenges for scientific data stewardship. 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.

The high demanding requirements on scientific data centers are reflected by the increasing popularity of cloud computing [3,4]. Cloud computing provides high performance and massive storage required for scientific applications in the same way, but with a lower infrastructure construction cost among many other features, because cloud computing systems are composed of data centers which can be clusters of commodity hardware datasets.

In this paper, we adopt Hadoop 2.0 as a principal component in our architecture to design the architecture of scientific data processing in the cloud. The rest of the paper is structured as follows. Section 2 discusses the related work. Section 3 introduces the proposed architecture and describes the implemented environment. Section 4 outlines the performances of our system through two experiments. Section 5 concludes the paper and describes the future research directions.

2. 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 in the cloud [5,6].

2.1 The MapReduce

MapReduce is considered as a high productivity alternative to traditional parallel programming paradigms for enterprise computing and scientific computing by removing the burden from programmer, such as tasks scheduling, fault tolerance, messaging, and data processing [7]. It was developed first by Google in 2004 as a parallel computing framework to perform distributed computing on a large number of commodity computers. MapReduce is a programming model for data processing. This programming model became popular because it is simple yet expressive enough to perform a large variety of computing tasks, from data mining to scientific computations.

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)$\rightarrow$\{key$_j$, value$_j$ | j=1$\cdots$k\}

Reduce:(key, [value$_1$, $\cdots$, value$_m$]) $\rightarrow$ (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, [value1, ..., valuem]). After receiving the parameters, Reduce is run to merge the data which were get from Map and output (key, final_value).

The map function 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 Reduce function 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]. Fig.1 shows a high level view of our architecture and how it processes the data.

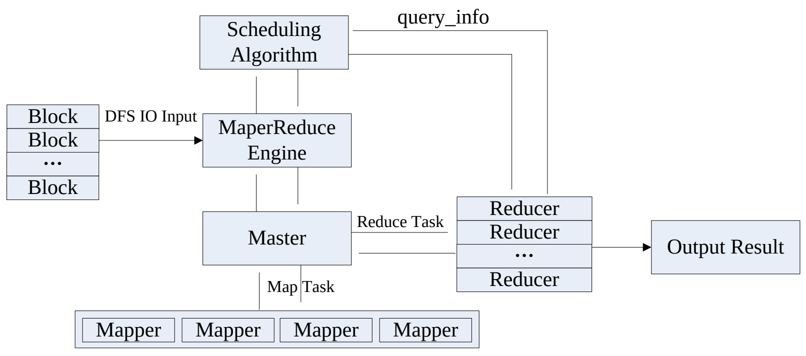

Fig.1. Mapreduce Working Flow

2.2 Hadoop 2.0

Hadoop which is an open source project and operates under the auspices of the Apache Software Foundation today was in production use at established and emerging web companies in 2006. 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 a master/slave architecture, and multiple replicas of data are stored on multiple compute nodes to provide reliable and rapid computations [11,12].

Hadoop2.0 which is the latest version has two major functionalities of the global ResourceManager (RM) and per-application ApplicationMaster (AM). RM has two main components: Scheduler and ApplicationsManager.

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.

The NodeManager is the per-machine framework agent who is responsible for launching the applications' containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the Scheduler. The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

3. SCIENTIFIC DATA PROCESSING USING HADOOP MAPREDUCE IN CLOUD ENVIRONMENTS

As show in Fig.2, 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.

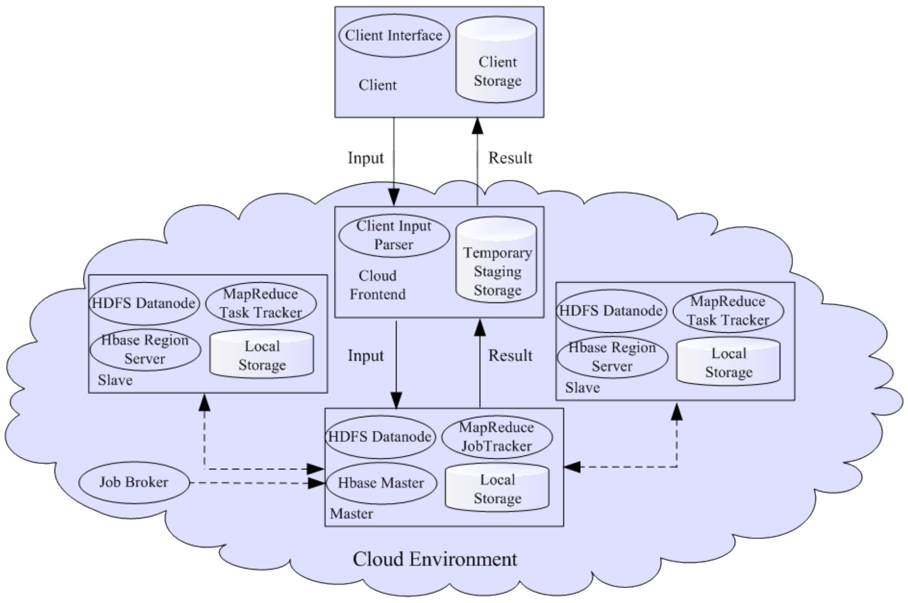

Fig.2. The Hadoop Architecture Based On Cloud Computing

3.1 Scientific Data Processor Using Hadoop MapReduce in Cloud Environments

Scientific data processor using Hadoop MapReduce in Cloud Environments consists of a front-end node called a JobTracker which is a simple master server and worker nodes called TaskTrackers which are slave servers. JobTracker is an interactive interface between users and the framework. When users submit the task to the JobTracker, JobTracker puts this task into the task queue and executes tasks according to the first come first served principle. JobTracker maintains the Map and Reduce tasks assigned to TaskTrackers [13]. TaskTracker executes instructions that get from JobTracker, and simultaneously deals with the exchange of data between Map and Reduce. Each node will periodically report the completed work and updated status to TaskTracker. If a TaskTracker doesn't communicate with JobTracker for a long time (it should be specified), JobTracker records this node as dead and assigns this node's data to other nodes.

The architecture is inspired by a batch of processing system, such as MapReduce. [14] and chosen a simple query semantics using map() and reduce() functions over the more widely used SQL-like approach. The resulting architecture allows for the seamless addition and removal of resources in a cloud environment. The map function takes a (key, value) pair and produces a list of pairs in a different domain. These tuples are then grouped under the same key. The resulting tuples of (key, [value]) pairs are processed by the reduce function that produces output values in a (possibly) different domain.

3.2 Scientific Data Processing Algorithm on MapReduce

The messages 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. 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. The master code would thus look like listed below.

```
master (){
    foreach slave {
        index , value = get_next_index_value_pair ();
        send (( index , value ), slave )
    }
    while ( work_available )
    {
        result , slave = receive_message_from_any_slave ();
        index , value = get_next_index_value_pair ();
        send (( index , value ), slave );
        output [ result . index ] = result . value ;
    }
    foreach slave {
        result , slave = receive_message_from_any_slave ();
        halt ( slave );
        output [ result . index ] = result . value ;
    }
}
```

The slave code would thus look like listed below.

```
slave (){
    until ( halted ){
    work = receive_message_from_master ();
    send (( work . index , work . value * work . value ), master );
    }
}
```

4. RESULTS

We evaluate how sequence alignment algorithms perform in a cloud by executing master and slave algorithm in a variety of configurations. Preliminary testing was done in a cluster of 4 nodes with single core processor, of which 3 nodes were used as the tasktrackers where the actual computations in Hadoop take place. The nodes contained 90GB of hard disk space and 2GB of physical memory and 6GB of virtual memory. Apache Hadoop2.0 was setup with 2 map tasks per node.

The web user interface of HADOOP framework contains the data related to the job that ran in the framework. The raw data of time consumed can be visualized as the graph shown in Fig.3. Fig.3 shows the pattern of execution of the Mapper job of the Reduce job, which contains the Reduce job, itself and the filter alignment job. TheReduce job aligns the sequences and the filter alignment job filters the aligned sequences according to the output requirement.

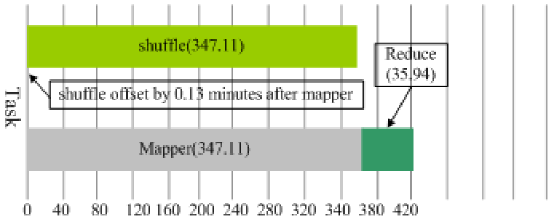

Fig.3 Timeline Plot For The Mapper And Reduce Job

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented our ongoing work towards creating a Hadoop MapReduce architecture that allows users to take advantage of resources from scientific clouds. The architecture takes care of provisioning Hadoop clusters and submitting jobs, allowing users to focus on writing their MapReduce application rather than managing cloud scientific resources.

In future work, we envise to deploying more frameworks, such as MPI, Twister… and so on, over Hadoop 2.0 platform, and examining their efficiency through the implementation of complex algorithms using greater numbers of nodes. Also, we plan to extend the proposed architecture to use Public Cloud resources from providers like AWS or Google App Engine.

REFRENCES:

[1] Xiao Liu, "Key Research Issues in Scientific Workflow Temporal Verification," THE FIRST CS3 PHD SYMPOSIUM 2010, pp. 49-51, 2010.

[2] Robert R. Downs and Robert S. Chen, "Self-Assessment of a Long-Term Archive for Interdisciplinary Scientific Data as a Trustworthy Digital Repository," JoDI: Journal of Digital Information, Vol. 11, 2010.

[3] Chao Jin and Rajkumar Buyya, "MapReduce Programming Model for .NET-based Distributed Computing," Technical report, The

University of Melbourne, Australia, October 2008.

[4] Sangmi Lee Pallickara, Shrideep Pallickara and Marlon Pierce, "Scientific Data Management in the Cloud: A Survey of Technologies, Approaches and Challenges," Handbook of Cloud Computing Springer Science+Business Media, 2010, pp. 517-527.

[5] R. Campbell, I. Gupta, M. Heath, S. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H. Lee and M. Lyons, "Open CirrusTM Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research," In USENIX Workshop on Hot Topics in Cloud Computing, 2009, pp. 1-1.

[6] H. Yang, A. Dasdan, R. Hsiao, and S. Parker, "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters," In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2007, pp. 1029–1040.

[7] Bill Howe, Peter Lawson, Renee Bellinger, Erik W. Anderson, Emanuele Santos, Juliana Freire, Carlos Eduardo Scheidegger, Antonio Baptista, and Claudio T. Silva, "End-to-End eScience: Integrating Workflow, Query, Visualization, and Provenance at an Ocean Observatory," In eScience '08: Proceedings of the 4th IEEE International Conference on eScience, 2008, pp. 127–134.

[8] Zhifeng Xiao and Yang Xiao, "Accountable MapReduce in Cloud Computing," The First International Workshop on Security in Computers, Networking and Communications, 2011, pp.1099-1104.

[9] Jaliya Ekanayake and Shrideep Pallickara, "MapReduce for Data Intensive Scientific Analyses," Fourth IEEE International Conference on eScience , 2008, pp. 277-284.

[10] Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun, "Map-Reduce for machine learning on multicore," In Advances in Neural Information Processing Systems 19 (NIPS 2006), 2006, pp. 281–288.

[11] Sangwon Seo, Ingook Jang1, Kyungchang Woo, Inkyo Kim, Jin-Soo Kim,and Seungryoul Maeng, "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment," In Proceedings of the 2009 IEEE Cluster, 2009, pp.1−8.

[12] B.Thirumala Rao and L.S.S.Reddy, "28 Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments," International Journal of Computer Applications, vol.34, 2011, pp. 28-32.

[13] DING Jian-li and YANG Bo, "A New Model of Search Engine based on Cloud Computing," International Journal of Digital Content Technology and its Applications. vol. 5, 2011, pp.236-243.

[14] W. Kleiminger, E. Kalyvianaki, and P. Pietzuch, "Balancing load in stream processing with the cloud," In Proceedings of the 6th International Workshop on Self Managing Database Systems, 2011, pp. 16-21.