

<http://ansinet.com/itj>

ITJ

ISSN 1812-5638

# INFORMATION TECHNOLOGY JOURNAL

**ANSI***net*

Asian Network for Scientific Information  
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

## Scientific Data Processing Using Mapreduce in Cloud Environments

Kong Xiangsheng

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

**Abstract:** This study focuses on the scientific data processing and transmission in map tasks. Based on Hadoop MapReduce of cloud computing, we propose the basic idea of scientific data prefetching mechanism proposed in this paper is to overlap the data transmission process with the scientific data processing process. And 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 study.

**Key words:** Map reduce, cloud computing, hadoop, distributed file system

### INTRODUCTION

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 (Liu, 2010). 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 (Downs and Chen, 2010). 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 also reflected by the increasing popularity of cloud computing (Jin and Buyya, 2008). 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 (Sangmi *et al.*, 2010). There are several vendors that offer cloud computing platforms, representative systems for cloud computing include Google App Engine, Amazon Elastic Compute Cloud (EC2), ATandT'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 (Campbell *et al.*, 2009). 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 study that Cloud Computing and Hadoop MapReduce are better approaches and solutions for scientific data processing.

### MAPREDUCE AND HADOOP

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 are a popular application on the Cloud (Yang *et al.*, 2007).

**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 (Howe *et al.*, 2008). 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 (Xiao and Xiao, 2011). MapReduce functions are as follows:

- Map:(in\_key,in\_value) $\rightarrow$  {key<sub>j</sub>, value<sub>j</sub> | j = 1...k}
- Reduce:(key, [value<sub>1</sub>,..., value<sub>m</sub>]) $\rightarrow$ (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, [value<sub>1</sub>, ..., value<sub>m</sub>]). 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 (Ekanayake *et al.*, 2008). 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 (Chu *et al.*, 2006). Fig. 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 (Seo *et al.*, 2009).

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 invoke 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.

**Hadoop:** Hadoop stores the intermediate results of the computations in local disks, where the computation tasks are run and then informs 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 (Thirumala Rao and Reddy, 2011). As shown in Fig. 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

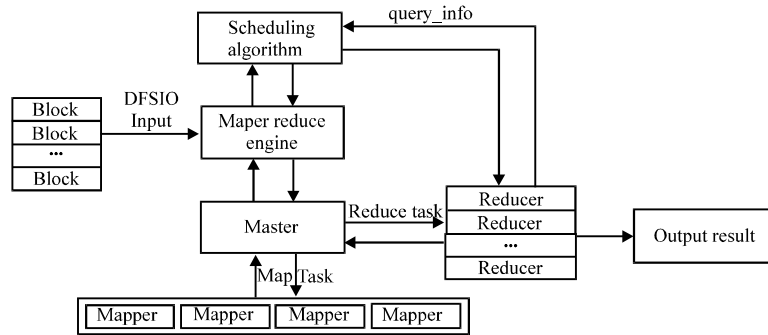


Fig. 1: MapReduce working flow

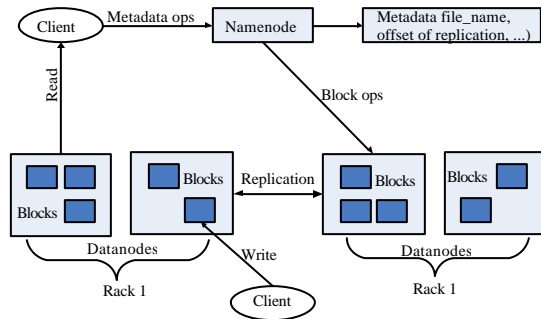


Fig. 2: HDFS Architecture

(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.

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.

## SCIENTIFIC DATA PROCESSING USING HADOOP 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.

**Parallel computing over clouds:** MapReduce is a style of parallel programming supported by capacity-on-demand clouds (Ding and Yang, 2011). 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  $p_{i,1}, p_{i,2}, p_{i,3}, \dots$  and that a Web page  $p_i$  contains words (terms)  $w_{j,1}, w_{j,2}, w_{j,3}, \dots$ . A basic but important structure in information retrieval is an inverted index, that is, a list:

$$(w_1; p_{1,1}, p_{1,2}, p_{1,3}, \dots)$$

$$(w_2; p_{2,1}, p_{2,2}, p_{2,3}, \dots)$$

$$(w_3; p_{3,1}, p_{3,2}, p_{3,3}, \dots),$$

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.

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  $p_i$  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 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 (Kleiminger *et al.*, 2011). 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.

### 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 3 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 4 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. Figure 4 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

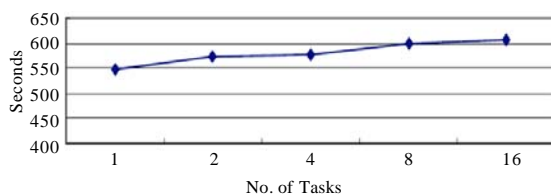


Fig.3: Test for scheduling overhead

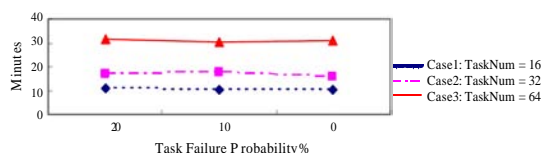


Fig. 4: Test for the degree of fault tolerance

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.

### CONCLUSIONS AND FUTURE WORK

As cloud computing is such a fast growing market, different cloud 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 and 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

- Campbell, R., I. Gupta, M. Heath, S. Ko and M. Kozuch *et al.*, 2009. Open Cirrus™ cloud computing testbed: Federated data centers for open source systems and services research. Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing, June 15, 2009, San Diego, CA., USA., pp: 1-5.
- Chu, C.T., S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng and K. Olukotun, 2006. Map-reduce for machine learning on multicore. Proceedings of the 19th Conference on advances Neural Information Processing Systems, December 4-7, 2006, Cambridge, MA., USA., pp: 281-288.
- Ding, J.L. and B. Yang, 2011. A new model of search engine based on cloud computing. Int. J. Digital Content Technol. Appl., 5: 236-243.
- Downs, R.R. and R.S. Chen, 2010. Self-assessment of a long-term archive for interdisciplinary scientific data as a trustworthy digital repository. J. Digital Inform., Vol. 11, No. 1.

- Ekanayake, J., S. Pallickara and G. Fox, 2008. MapReduce for data intensive scientific analyses. Proceedings of the IEEE 4th International Conference on eScience, December 7-12, 2008, Indianapolis, IN., USA., pp: 277-284.
- Howe, B., P. Lawson, R. Bellinger, E. Anderson and E. Santos *et al.*, 2008. End-to-end eScience: Integrating workflow, query, visualization and provenance at an ocean observatory. Proceedings of the 4th IEEE International Conference on eScience, December 7-12, 2008, Indianapolis, IN., USA., pp: 127-134.
- Jin, C. and R. Buyya, 2008. MapReduce programming model for.NET-based distributed computing. Technical Report, The University of Melbourne, Australia. <http://gridbus.cs.mu.oz.au/reports/MapReduce-NET-2008.pdf>
- Kleiminger, W., E. Kalyvianaki and P. Pietzuch, 2011. Balancing load in stream processing with the cloud. Proceedings of the IEEE 27th International Conference on Data Engineering Workshops, April 11-16, 2011, Hanover, Germany, pp: 16-21.
- Liu, X., 2010. Key research issues in scientific workflow temporal verification. Proceedings of the 1st CS3 PhD Symposium, February 26, 2010, Australia, pp: 49-51.
- Sangmi, L.P., S. Pallickara and M. Pierce, 2010. Scientific Data Management in the Cloud: A Survey of Technologies, Approaches and Challenges. In: Handbook of Cloud Computing, Furht, B. and A. Escalante (Eds.). Springer, New York, USA., ISBN: 9781441965233, pp: 517-533.
- Seo, S., I. Jang, K. Woo, I. Kim, J.S. Kim and S. Maeng, 2009. HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment. Proceedings of the IEEE International Conference on Cluster Computing and Workshops, August 31-September 4, 2009, New Orleans, LA., USA., pp: 1-8.
- Thirumala Rao, B. and L.S.S. Reddy, 2011. Survey on improved scheduling in hadoop mapreduce in cloud environments. *Int. J. Comput. Appl.*, 34: 29-33.
- Xiao, Z.F. and Y. Xiao, 2011. Accountable MapReduce in cloud computing. Proceedings of the IEEE Conference on Computer Communications Workshops, April 10-15, 2011, Shanghai, China, pp: 1082-1087.
- Yang, H.C., A. Dasdan, R.L. Hsiao and D.S. Parker, 2007. Map-reduce-merge: Simplified relational data processing on large clusters. Proceedings of the ACM SIGMOD International Conference on Management of Data, June 11-14, 2007, Beijing, China, pp: 1029-1040.