IJERST

International Journal of Engineering Research and Science & Technology

ISSN : 2319-5991
Vol. 4, No. 3
August 2015

www.ijerst.com

Email: editorijerst@gmail.com or editor@ijerst.com

International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
Vol. 4, No. 3, August 2015
© 2015 IJERST. All Rights Reserved

Research Paper

MAP REDUCE PROGRAMMING MULTI-CLOUDS WITH BSTREAM BASED ON HADOOP

K Suganya[1*] and S Dhivya[1]

Corresponding Author: **K Suganya ✉ suganindiramca@gmail.com*

In Cloud Computing is having huge concentration and helpful to inspect large amounts of datasets. Execution of MapReduce code in cloud has a big difficulty of optimization of resource to reduce the profitable cost or job finishing time for individual jobs, power supervision and performance. MapReduce works on bulk dataset which contains various type of in order and computation, MapReduce translator provides a sufficient result to help simply deploy an application on cloud system through transforming sequent codes. In current days, different SQL to-MapReduce translate similar like-SQL queries to MapReduce order also have a greater performance on cloud computation. Cloud MapReduce is commonly used because more accomplished and execution will rapidly than other services of the MapReduce framework. BStream is cloud framework for MapReduce is combine stream processing in external cloud with Adopt in Internal Cloud (IC), it uses stream processing engine.

Keywords: Hadoop, MapReduce, Multi-cloud, HDFS

INTRODUCTION

Cloud is an execution environment of resources containing of multiple stakeholders and providing a service at multiple ways for a specified level of quality to be more specific, like cloud is an infrastructure that enables execution of (some type of services and applications).The process of stream in big-data is real-time parallelized processing frameworks, like a Hadoop MapReduce (Vecchiola, 2012), the major difficulty for their designs are first and foremost artificial to access and process the static input data. The iterative module is able to use when the input data appear in a stream flow. Moreover, the existing system is unable to control when the streaming in coming datasets are from different sources and have different entry rates.

Three basic types of cloud services.

INFRASTRUCTURE AS A SERVICE

IaaS is the Ist layer and foundation of the cloud computing. Using type service, you control all your applications, information, OS, middleware and etc. The service supplier Controls your VM, all servers, networking process and storage disk. This service allows you to avoid on hardware and human interactions, reduce your risk, and

[1] Department of MCA, V.S.B Engineering College, Karur, Tamil Nadu, India.

streamline and automate scaling. Some of the top names in IaaS like VM Ware, Microsoft, Amazon, Red Hat and Rack space.

Platform as a Service

This type of service model could be considered as the 2nd layer. You manage your applications and information and the cloud dealer manages everything else. Benefits for using PaaS include efficient version deployment and the ability to change or upgrade and minimize expenses. One popular Platform-as-a-Service is the Google app engine.

Software as a Service

This is the last layer of the cloud service. This agrees to your business to run agendas in the cloud where all segments are managing by the cloud dealer. Your clients will have guaranteed because all are using the same software's. Example of this is online banking job and email sharing such as Gmail and Hotmail.

RELATED WORK

This thesis proposes a task-level adaptive Map Reduce structure to process streaming records in scientific applications. This structure extends the established Hadoop Map Reduce framework and also specifically addresses the different entry rate of data openings. The structure is developed to scale in various cloud platforms by applying scaling theorems and scaling corollaries. This is now fully designed and distribute as a standalone simulator.

MapReduce subtraction consists of mapping phase, shuffling and sorting phase, followed by reducing phase. Map process takes a key value pairs as input and produces a list of intermediate key value pairs that are divider and stored locally on each data. In the shuffle phase, reducers fetch the intermediate data corresponding to their partition from all the mappers. The sorting phase, they sort the shuffled data according to the key, to group values for the same key. Finally, they are applying the reducing operation on the list of values for each separate key to produce one or more key value pairs. MapReduce runtime parallelizes the effecting of these functions and handles issues related to load balancing and fault tolerance.

Hadoop, an open source completion of MapReduce, consists of like Hadoop Distributed File System (HDFS) and MapReduce runtime. The input data's for MapReduce trades are dividing into fixed size (default 64 MB) blocks and stored in HDFS. This MapReduce runtime follows master worker architecture. The master is Job Tracker to assigns tasks to the every worker nodes. Each worker node runs on a Task Tracker which manages the currently allocated task. Each worker node is organizing with a fixed number of mapping and reducing slots. This hard partitioning of resources leads to under-utilization and wastage of compute cycles.

EXISTING TECHNOLOGIES

Cloud bursting provides an alternative to over provisioning by offloading the excess load from the Internal Cloud (IC) to an External Cloud (EC) (like Rack space, Amazon EC2). However, there are two main difficulties in scaling MapReduce using cloud bursting. First, being a batch-processing system requires the entire input data for the job to be materialized before the start of computation. As shuffle involves all-all node communication, shuffling of data from EC to reducers in IC takes longer due to difference between inter cloud and intra-cloud bandwidth. This elongated shuffle phase not only delays job completion but also causes idle CPU cycles within reducers.

PROPOSED TECHNOLOGIES

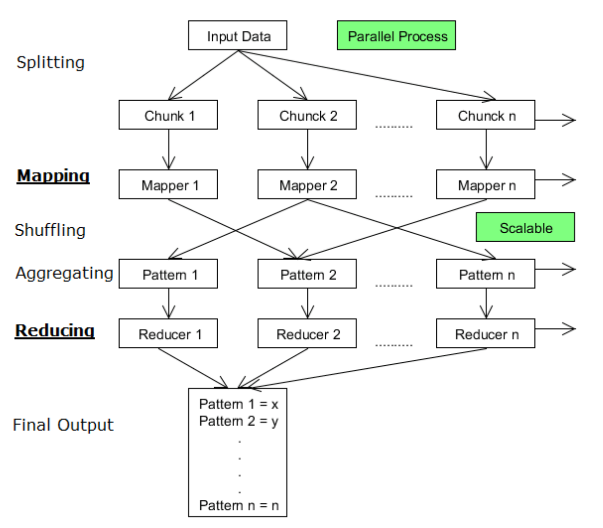
These thesis works propose BStream, a new cloud bursting framework to address the above difficulties. BStream uses stream processing engine called EC and IC. Using both map and reduce process execute on the incoming stream of data as and when it entering in EC. This overlies processing with input data transfer, thus Minimizing startup latencies in EC. Shuffle from EC is also decoupled from reducers, allowing the reducers to start much later in the job lifecycle. Shuffle is further optimized using check pointing strategies that download intermittent reduce output from EC to reducers in IC. Thus, using stream processing and check pointing strategies, BStream enables pipelined uploading, processing and downloading of data. BStream uses an analytical model to estimate what portions of MapReduce job to burst, when to burst and when to start the reducers, to meet job deadline. We currently consider meeting deadlines for individual jobs whose input data is initially present only in IC. BStream in big data is also quite interesting as it provides a mechanism to manage the uncertainty that is inherent in this scenario because of the variety and veracity of data.

ARCHITECTURE

In the above MapReduce process flow is given below:

- The input data's can be splitting into n number of chunks depending upon the amount of data's and processing capability of individual unit.
- Next, it is passed to the mapping functions. Please note that all the chunks are executed simultaneously at the same time, which embraces the parallel processing of data's.
- After that, shuffling happens which leads to aggregation of similar patterns.
- Finally, reducers merge them all to get a combined output as per the logic.
- This algorithm embraces scalability as depending on the size of the input data, we can keep increasing the number of the parallel processing units.

Figure 1: Architecture for Map Reduce



EXAMPLE IMPLEMENTATION

In our word count implementation, want to count the number of word occasion so that can get count on word frequencies. Thus, want our reducing script to simply sum the *values* of the group of <key, value> pairs which have the equal key.

When write a MapReduce process flow, we'll have to write two scripts: the mapping script, and the reducing script. The rest will be managed by the Amazon EMR (like Elastic MapReduce) framework.

When start a mapping/reducing process flow, the framework will *divide* the input into cracks, passing each segment to a various machine. Every machine then runs the *mapping script* on the portion of data attributed to it.

The *mapping script* takes some input data's, and maps it to <key, value> pairs according to your conditions. For example, if we wanted to count word frequencies in a text, have <word, count> be our <key, value> pairs.

MapReduce then would emit a <word, 1> pair for each word in the input stream. Note that the map script does no *aggregation* (i.e., actual counting) this is what the reduce script it for. The purpose of the map script is to model the data into <key, value> pairs for the reducer to aggregate.

Emitted <key, value> pairs are then "shuffled" (to use the terminology in the diagram below), which basically means that pairs with the same key are grouped and passed to a single machine, which will then run the *reduce script* over them. The *reduce script* (which you also write) takes a collection of <key, value> pairs and "reduces" them according to the user-specified reduce script.

Word count Mapper

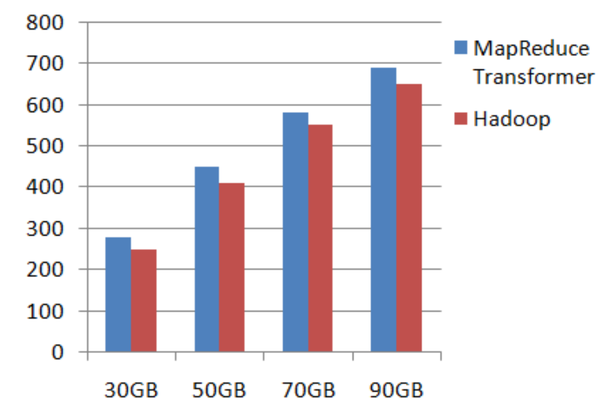
```
Public static class Map extends MapReduce
Mapper<LongWrite, Text, Text,
IntWrite>
{
Private static IntWrite one = new IntWrite(1);
Private Text words = new Text();
Public static String map(Long key, Text value,
OpCollector<Text, IntWrite> output,
Reporter reporter) throws IOException {
String line = value.toString();
StringTokenizer itr = new StringTokenizer(line);
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
output.collect(word, one);
}}}
```

Word count Reducer

```
public static class Reduce extends ReduceBase
implements Reducer<IntWrite, Text,
IntWritable>
{
Public static void reduce(Text,
Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException {
int sum = 0;
while (values.hasNext()) {
sum += values.next().get();
}
output.collect(key, new IntWritable(sum));
}
}
```

Comparison Chart

Figure 2: Word Count Application Results



CONCLUSION AND FUTURE WORK

The emergence of Cloud and general increase in the importance of data intensive applications, programming models for data intensive application have increased significant attention. In this paper, presented BStream with Hadoop to multiple clouds by combining stream processing of EC with batch processing in IC. BStream uses an analytical model to estimate resource allocation and task allocation across clouds. Showed that the performance of analytical model is reasonably accurate. Compared the performance of BStream with other existing works and deals that stream processing along with continuous check pointing in EC can significantly improve performance. Finally, characterized the operational regime of BStream, concreting way for meeting time limit with multiple jobs.

Future programming frameworks must allow client systems to expand robust, scalable programming models that, while relying on equivalent computation, abstract the details of parallelism. The end programmer should be showing only with a set of applications rather than the details of the distributed hardware.

REFERENCES

1. Apache Hadoop. http://hadoop.apache. org.
2. Chang F *et al.* (2008), "Bigtable: A distributed storage system for structured data", *ACM Transactions on Computer Systems* (TOCS), Vol. 26, No. 2, pp. 1-26.
3. Chris Miceli *et al.* (2009), Programming Abstractions for Data Intensive Computing on Clouds and Grids, 9th IEEE/ACM International Symposium on Cluster Computing and the Grid.
4. Condie T, Conway N, Alvaro P, Hellerstein J M, Elmeleegy K and Sears R (2010), "MapReduce online," in Proc. 7th USENIX Conf. Netw. Syst. Des. Implementation, p. 21.
5. Costa F, Veiga L, and Ferreira P (2013), "Internet-scale support for map-reduce processing," *J. Internet Serv. Appl.,* Vol. 4, No. 1, pp. 1-17, 2013.
6. Dean J *et al.* (2010), "MapReduce: a flexible data processing tool", *Communications of the ACM,* Vol. 53, No. 1, pp. 72-77.
7. Dean J and Ghemawat S (2004), "Mapreduce: Simplified data processing on large clusters. In OSDI'04", Sixth Symposium on Operating System Design and Implementation (December 2004).
8. Ekanayake J *et al*. (2008), "Mapreduce for data intensive scientific analyses", *In the 4th IEEE International Conference on eScience,* pp. 277-284.
9. Jahani E *et al*. (2011), "Automatic Optimization for MapReduce Programs", *Proceedings of the VLDB,* Vol. 4, No. 6, pp. 385-396.
10. Kc K and Anyanwu K (2010), "Scheduling hadoop jobs to meet deadlines," *in Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, pp. 388-392.
11. Kim H and Parashar M (2011), "Cometcloud: An autonomic cloud engine," in Cloud Computing: Principles and Paradigms, Hoboken, NJ, USA: Wiley, 2011, Ch. 10, pp. 275-297.

12. Kim H, Chaudhari S, Parashar M, and Marty C (2009), "Online risk analytics on the cloud," *in Proc. IEEE/ACM 9th Int. Symp. Cluster Comput. Grid*, pp. 484-489.

13. Morton K *et al*. (2010), "Paratimer: a progress indicator for mapreduce dags", *In Proceedings of the 2010 ACM SIGMOD*, pp. 507-518.

14. Olston C, Chiou G, Chitnis L, Liu F, Han Y, Larsson M, Neumann A, Rao V B, Sankarasubramanian V, Seth S, Tian C, ZiCornell T, and Wang W (2011), "Nova: continuous Pig/Hadoop workflows," *in Proc. ACM Int. Conf. Manage. Data*, pp. 1081-1090.

15. Pike R *et al*. (2005), *Interpreting the data: Parallel analysis with Sawzall. Scientific Programming,* Vol. 13, No. 4, pp. 277-298.

16. Shantenu Jha, Daniel S Katz, Andre Luckow, Andre Merzky, Katerina Stamou (2009), *Understanding Scientific Applications For Cloud Environments.*

17. Vecchiola C, Calheiros R N, Karunamoorthy D, and Buyya R (2012), "Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka," *Future Generation Comput. Syst*., Vol. 28, No. 1, pp. 58–65.

ISSN 2319-5991