Elysium Journal
Engineering Research & Management

P-ISSN: 2347-4408
E-ISSN: 2347-4734

GEO-DISTRIBUTED MAP REDUCE FRAMEWORK FOR COST EFFICIENT BIG DATA ANALYSIS

RA Nagarajan and A Muthumari

Research & Development, Elysium Technologies Private Limited, Madurai, Tamil Nadu, India

Abstract: ***Big data analysis is one of the major challenges of current era. The limits to what can be done are often times due to how much data can be processed in a given time-frame Implementation of map reduce framework in Hadoop plays an important role in handling and processing big data. Hence we concentrate on geographical distribution of geo-distributed data for sequential execution of map reduce jobs to optimize the execution time. Our paper introduces Location based job execution system, a system for efficiently processing geo-distributed big data. Geo-MapReduce is a Hadoop based framework that can efficiently perform a sequence of MapReduce jobs on a geo-distributed dataset across multiple datacenters. It act much like the atmosphere surrounding the clouds. The problem of executing geo-distributed MapReduce job sequences as arising in "cloud-of-clouds" scenarios is analyzed for job execution. For distributing the input data, DTG algorithm is applied to identify optimized execution path. The datacenter with optimized execution path is selected for job execution. The execution paths for performing a MapReduce job on a geo-distributed dataset are copy, geo and multi. Our system uses Multi execution method for efficient and optimized execution.***

Keywords: Geographical Distributed Servers, MapReduce, Hadoop, Big Data Analytics, Cost Efficient.

1. INTRODUCTION

Big data analysis is one of the major challenges of our era. The limits to what can be done are often times due to how much data can be processed in a given time-frame. Big datasets inherently arise due to applications generating and retaining more information to improve operation, monitoring, or auditing; applications such as social networks support individual users in generating increasing amounts of data. Implementations of the popular MapReduce framework [1], such as Apache Hadoop [2], have become part of the standard toolkit for processing large datasets using cloud properties, and are provided by most cloud vendors. In short, MapReduce works by dividing input files into chunks and processing these in a series of parallelizable steps. As suggested by the name, mapping and reducing constitute the essential phases for a MapReduce job. I n the former phase, mappers processes read respective input file chunks and produce key , Val pairs called intermediate data. Each reducer process atomically applies the reduction function to all values of each key assigned to it.

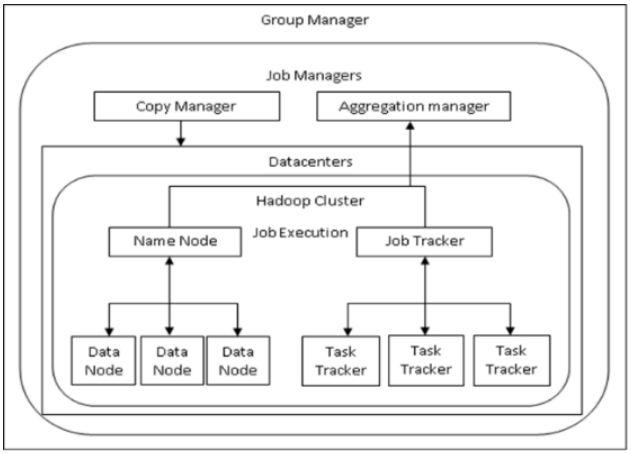

Figure 1 Hadoop Proposed Cluster Architecture

As the initial hype of cloud computing is wearing off, users are starting to see beyond the illusion of Omni-present computing resources and realize that these are implemented by concrete data centers, whose locations matter. More and more applications relying on cloud platforms are geo-distributed, for any (combination) of the following reasons:

- Data is stored near its respective sources or frequently accessing entities (e.g., clients) which can be distributed, but the data is analyzed globally
- Data is gathered and stored by different (sub-)organizations, yet shared towards a common goal
- Data is replicated across datacenters for availability, incompletely to limit the overhead of costly updates.

To make matters worse, MapReduce jobs do not always come alone. Frequently, sequences of MapReduce jobs are executed on a given input by applying the first job on the given input, applying the second job on the output of the first job, and so on. An example is the handling of large Web caches by executing an algorithm such as PageRank [10]. This algorithm constructs a graph that describes the inter-page relationships, which are refined using further MapReduce jobs. Another

P-ISSN: 2347-4408
E-ISSN: 2347-4734

example is hash-tree generation, performed to verify the integrity of data. Each level of the hash-tree can be generated using a MapReduce job starting from leaves representing the original input. Many times distinct MapReduce jobs are applied in sequence rather than iteratively performing the same job. For instance the query performed on the geo-distributed Web cache mentioned above may have a filtering step and a content search step, which need to be executed as two consecutive jobs.

2. BACKGROUND AND RELATED WORK

Hadoop is a Java-based MapReduce implementation for large clusters. It is bundled with the Hadoop Distributed File System (HDFS), which is optimized for batch workloads such as those of MapReduce. In many Hadoop applications, HDFS is used to store the input of the map phase as well as the output of the reduce phase.

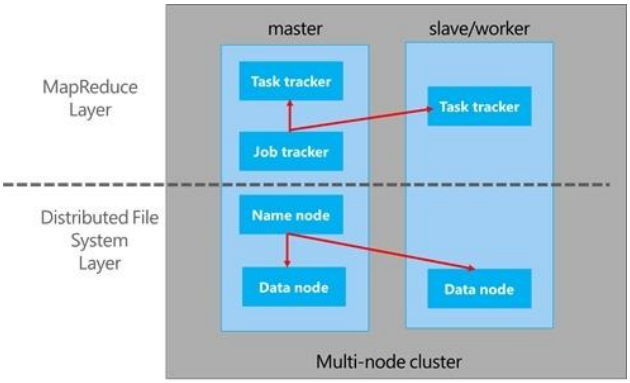

Figure 2 Multi Node Cluster

HDFS is, however, not used to store intermediate results such as the output of the map phase. They are stored on the individual local file systems of nodes. The Hadoop follows a master-slave model where the master is implemented in Hadoop's Job Tracker. The master is responsible for accepting jobs, dividing those into tasks which encompass mappers or reducers, and assigning those tasks to slave worker nodes. Each worker node runs a Task Tracker that manages its assigned tasks. A default split in Hadoop contains one HDFS block (64 MB), and the number of file blocks in the input data isused to determine the number of mappers.

A new scheduling algorithm, Longest Approximate Time to End (LATE) is designed, that is highly robust to heterogeneity. we address the problem of how to robustly perform speculative execution to maximize performance. Hadoop's scheduler starts speculative tasks based on a simple heuristic comparing each task's progress to the average progress. Although this heuristic works well in homogeneous environments where stragglers are obvious, we show that it can lead to severe performance degradation when its underlying assumptions are broken. The goal of speculative execution is to minimize a job's response time. Response time is most important for short jobs where a user wants an answer quickly, such as queries on log data for debugging, monitoring and business intelligence. We always speculatively execute the task that we think will finish farthest into the future, because this task provides the greatest opportunity for a speculative copy to overtake the original and reduce the job's response time.

As cloud services grow to span more and more globally distributed datacenters, there is an increasingly urgent need for automated mechanisms to place application data across these datacenters. We present Volley, a system that addresses these challenges. Cloud services make use of Volley by submitting logs of datacenter requests. Volley analyzes the logs using an iterative optimization algorithm based on data access patterns and client locations, and outputs migration recommendations back to the cloud service. To utilize Volley, applications have to log information on the requests they process. These logs must enable correlating requests into "call trees" or "runtime paths" that capture the logical flow of control across components, as in Pinpoint [7] or X-Trace [14]. For the Live Mesh and Live Messenger commercial cloud services, the data volumes from generating Volley logs are much less than the data volumes from processing user requests. Once the data is in Cosmos, Volley periodically analyzes it for migration opportunities. To perform the analysis, Volley relies on the SCOPE [5] distributed execution infrastructure, which at a high level resembles MapReduce [11] with a SQL-like query language. In our current implementation, Volley takes approximately 14 hours to run through one month's worth of log files. We first map each client to a set of geographic coordinates using the commercial geo-location database mentioned earlier. Iteratively Move Data to Reduce Latency. Volley iteratively moves data items closer to both clients and to the other data items that they communicate with. This iterative update step incorporates two earlier ideas: a weighted spring model as in Vivaldi [9] and spherical coordinates as in Htrae [36]. After computing a nearly ideal placement of the data items on the surface of the earth, we have to modify this placement so that the data items are located in datacenters, and the set of items in each datacenter satisfies its capacity constraints. Like Phase 2, this is done iteratively: initially, every data item is mapped to its closest datacenter. For datacenters that are over their capacity, Volley identifies the items that experience the fewest accesses, and moves all of them to the next closest datacenter. Because this may still exceed the total capacity of some datacenter due to new additions, Volley repeats the process until no datacenter is over capacity.

Disk-oriented approaches to online storage are becoming increasingly problematic: they do not scale grace-fully to meet the needs of large-scale Web applications, and improvements in disk capacity have far out-stripped improvements in access latency and bandwidth. In this paper we argue that a new class

P-ISSN: 2347-4408
E-ISSN: 2347-4734

of storage called RAMCloud will provide the storage substrate for many future applications.

A RAM Cloud stores all of its in-formation in the main memories of commodity servers, using hundreds or thousands of such servers to create a large-scale storage system. Because all data is in DRAM at all times, a RAM Cloud can provide 100-1000x lower latency than disk-based systems and 100-1000x greater throughput. RAM Clouds are most likely to be used in datacenters containing large numbers of servers divided roughly into two categories: application servers, which implement application logic such as generating Web pages or enforcing business rules, and storage servers, which provide longer-term shared storage for the application servers.

A single large RAM Cloud system must support shared access by hundreds of applications of varying sizes. we have argued that the best long-term solution for many applications may be a radical approach where all data is kept in DRAM all the time. The two most important aspects of RAM Clouds are (a) their extremely low latency and (b) their ability to aggregate the resources of large numbers of commodity servers. Together, these allow RAM Clouds to scale to meet the needs of the largest Web applications. In addition, low latency also enables richer query models, which will simplify application development and enable new kinds of applications. Finally, the federated approach makes RAM-Clouds an attractive substrate for cloud computing environments that require a flexible and scalable storage system.

3. EXISTING MAPREDUCE EXTENSIONS

There have been many efforts to improve the efficiency of the MapReduce job. Other storage systems like RAM Cloud can efficiently handle large amounts of data but have to be deployed within a datacenter. Other System includes a programming model and a framework for developing massively scalable distributed applications. Some system introduce approximation algorithms that order MapReduce jobs to minimize overall job completion time. We describe the model of geo-distributed systems, data, and operations as considered in this paper.

First define a partition size to keep our solution bounded. To move data from stage *s to next stage s + 1 a* MapReduce phase is applied to data partitions and the same number of (output) data partitions are created. Other storage systems can support geo-distributed big data in cloud environments. Walter is a key-value storage system that can hold massive amounts of geo distributed data. COPS is another storage system for geo-distributed data. Dryad includes a programing model and a framework for developing massively scalable distributed applications. The existing system do not address actual computations over stored data. It solve only the part of the problem. MapReduce makes no assumptions about the execution path. Hence, it offers only very limited optimization opportunities. Dryad provides the programmer with more fine grained control over the execution but does not possess mechanisms to process geo-distributed data.

4. GEO DISTRIBUTED SERVER SELECTION USING MAP-REDUCE

The term Web-Scale Data Management has been coined for describing the challenge to develop systems that scale to data volumes as they are found in search indexes, large scale warehouses, and scientific applications like climate research. Many new architectures have been suggested, among which the map/reduce paradigm and its open source implementation Hadoop have gained the most attention. Here, programs are written as map and reduce functions, which process key/value pairs and can be executed in many data parallel instances. The big advantage of that programming model is its generality: Any problem that can be expressed with those two functions can be executed by the framework in a massively parallel way. The map/reduce execution model has been proven to scale to 1000s of machines.

A Geo-distributed Map Reduce (G-MR) framework is proposed for efficient job execution with optimized resource consumption. GMR introduce solution for optimizing the execution of a MapReduce job sequence on a given dataset. G-MR uses the DTG algorithm to determine an optimized path to perform the sequence of MapReduce jobs and uses Hadoop MapReduce clusters deployed in each of the considered datacenters to execute the determined optimized path accordingly. Job Managers perform the jobs accordingly using the Hadoop MapReduce clusters deployed in corresponding datacenters. It supports fficient Job execution, applicable for different type of job and execution in optimized time consumption.

Methodologies and Implementation Design

4.1 Hadoop Installation

i. JDK installation

ii. Cygwin installation

iii. Environment Variable setup

iv. VM ware installation

v. Hadoop Installation

   a. Multi node – installation

vi. Eclipse for binding Hadoop with windows

4.2 Data Preprocessing

A Geo- distributed data called census data is taken as input for processing. The U.S. Census Bureau produces and publishes

P-ISSN: 2347-4408
E-ISSN: 2347-4734

estimates of the population for each state and county, as well as the nation as a whole. We utilize administrative data from a number of sources to estimate 1) the change in population since the most recent decennial census, and 2) the population for each year since the most recent decennial census. With each annual release of population estimates, the entire time series of estimates beginning on April 1, 2010 is revised and updated. This census dataset is used further. The dataset is analyzed and preprocessed to remove noise and unwanted fields from the dataset before applying into hadoop for execution.

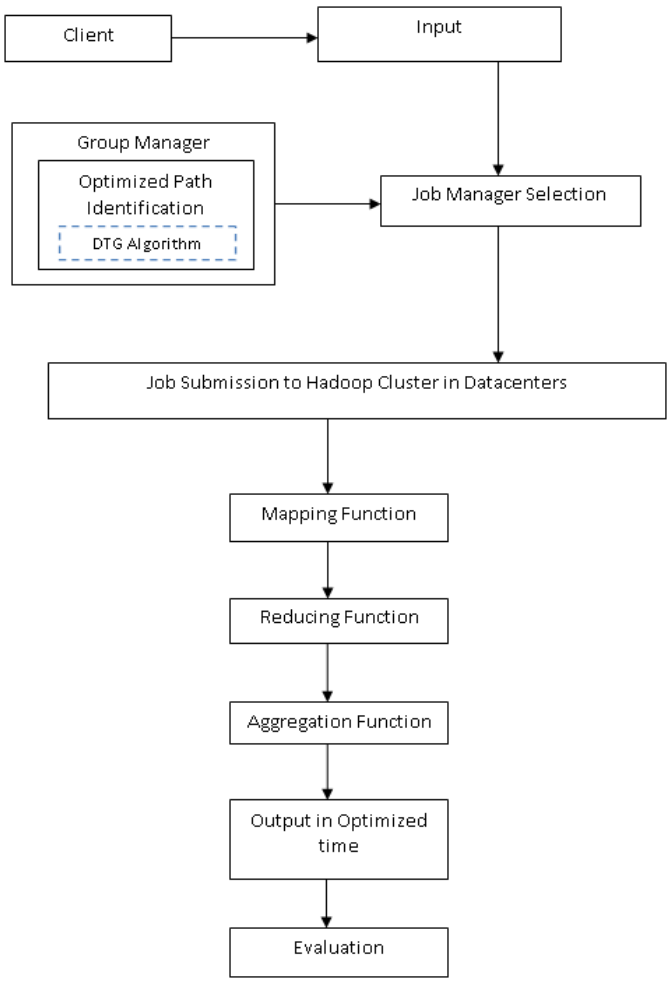

Figure 2 Proposed Flow Diagram

Noise data like invalid data, null values, etc are identified and removed from the dataset. Our target is fixed for job execution and based on the target (output) needed, fields will be selected for processing.

4.3 HDFS Upload

Hadoop Distributed File System (HDFS) serves as the large scale data storage system. Similar to other common file systems, the HDFS supports hierarchical file organization. The Name Node splits large files into fixed sized data blocks which are scattered across the cluster. Typically the data block size for the HDFS is configured as 128MB, but it can be configured by file system clients as per usage requirements. Hadoop Distributed File System (DFS) will be configured for uploading the preprocessed geo data into Hadoop. The configuration includes setting VM (Hadoop platform) IP and port for connection. FSDataInputStream and FSDataOutputStream is used to upload and download data from Hadoop. Different datacenters are analyzed for data execution across different datacenters.

4.4 Job Execution

The Group Manager executes the DTG algorithm (Novel Algorithm). This group manager is executed for determining path for performing the MapReduce job. After this the Group Manager starts to execute the job in determined optimized Multi-execution path based on the parallelization contract (PACT). Input contracts is defined based on the first order function which is the target that we need to achieve using the census dataset and MapReduce framework. The input contract includes the properties that we need to implement the second order function. The second order function is the function that we need to define based on the output data. Executing individual MapReduce jobs in each datacenter on corresponding inputs and then aggregating results is defined as MULTI execution path. This path and PACT programming is used to execute the jobs effectively.

4.5 Executing Jobs

Job Managers perform the jobs accordingly using the Hadoop MapReduce clusters deployed in corresponding datacenters. The Group-Manager may also instruct a Job Manager to copy data to a remote datacenter or aggregate multiple sub-datasets copied from two or more remote datacenters. The effective job execution can be achieved using the Geo-MapReduce and PACT Model. The job tracker forwards the task to task tracker for execution and the results from the task tracker is received and combined to get aggregative result.

5. CONCLUSION

This paper presents Geo-MapReduce, a MapReduce framework that can efficiently execute a sequence of MapReduce jobs on geo-distributed datasets. G-MR relies on a novel algorithm termed DTG algorithm which looks for the most suitable way to perform a job sequence, minimizing either execution time or cost. We illustrate through real MapReduce application scenarios that G-MR can substantially improve time or cost of job execution compared

P-ISSN: 2347-4408
E-ISSN: 2347-4734

to naıve schedules of currently widely followed deployments. We believe that our framework is also applicable to single datacenters with non-uniform transmission characteristics, such as datacenters divided into zones or other network architectures.

REFERENCE

1. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI, 2004.*
2. Apache Software Foundation, "Hadoop," http://hadoop.apache.org.
3. S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated Data Placement for Geo-Distributed Cloud Services," in *NSDI, 2010.*
4. "Data from Year 2000 US Census," http://aws.amazon.com/datasets/Economics/2290.
5. "Department of Defense Information Enterprise Strategic Plan 2011- 2012," http://dodcio.defense.gov/docs/DodIESP-r16.pdf.
6. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI, pages 137–150, 2004.
7. C. Olston, B. Reed, A. Silberstein, and U. Srivastava. Automatic Optimization of Parallel Dataflow Programs. In R. Isaacs and Y. Zhou, editors, USENIX Annual Technical Conference, pages 267–273. USENIX Association, 2008.
8. S. Fushimi, M. Kitsuregawa, and H. Tanaka. An Overview of The System Software of A Parallel Relational Database Machine GRACE. In W. W. Chu, G. Gardarin, S. Ohsuga, and Y. Kambayashi, editors, VLDB, pages 209–219. Morgan Kaufmann, 1986.
9. A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis. In U. ¸ Cetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, editors, SIGMOD Conference, pages 165–178. ACM, 2009.
10. P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access Path Selection in a Relational Database Management System. In P. A. Bernstein, editor, SIGMOD Conference, pages 23–34. ACM, 1979.