Volume 5, Issue 7, July 2015 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Configuration Tuning Parameters for Map Reduce Programming Model for Hadoop based Big Data Management

K.Sangeetha., M.Sc., M.Phil
SSNC College – Kanavaipudur
Computer Science Department
Periyar University, Salem,
Tamil Nadu, India

S.Praba., M.Sc., M.Phil
SSNC College – Kanavaipudur
Computer Science Department
Periyar University, Salem,
Tamil Nadu, India

M.Mujeeb., M.B.A (IT)., M.Phil
AVS Atrs & Science College
Management Department
Periyar University, Salem,
Tamil Nadu, India

Abstract: ***The concept of big data has been endemic within computer science since the earliest days of computing. "Big Data" originally meant the volume of data that could not be processed efficiently) by traditional database methods and tools. Big Data has three dimensions, volume, variety, and velocity. Big data technologies describe a new generation of technologies and architectures designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis. Hadoop is currently the most mature, accessible, and popular implementation of the MapReduce programming model. Hadoop is usually supported by the Hadoop Distributed File System (HDFS). In the programming model of MapRedue, the input of the computation is a set of key/value pairs, and the output is also a set of key/value pairs usually in a different domain from the input. Users define a map function which converts one input key/value pair to an arbitrary number of intermediate key/value pairs, and a reduce function which merges all intermediate values of the same intermediate key into a smaller set of values, typically one value for each intermediate key. In this paper we analyzed the Characteristic of Big Data, the basic structure of HDFS and Read/Write operations are performed. We also analyzed the various configuration parameters which affects the performance of the Map Reduce job run.***

Keywords: Big Data, HDFS, Map Reduce, Hadoop

I. INTRODUCTION

The concept of big data has been endemic within computer science since the earliest days of computing. "Big Data" originally meant the volume of data that could not be processed (efficiently) by traditional database methods and tools. Each time a new storage medium was invented, the amount of data accessible exploded because it could be easily accessed. The original definition focused on structured data, but most researchers and practitioners have come to realize that most of the world's information resides in massive, unstructured information, largely in the form of text and imagery. The explosion of data has not been accompanied by a corresponding new storage medium.

We define "Big Data" as the amount of data just beyond technology's capability to store, manage and process efficiently. These imitations are only discovered by a robust analysis of the data itself, explicit processing needs, and the capabilities of the tools (hardware, software, and methods) used to analyze it.

As with any new problem, the conclusion of how to proceed may lead to a recommendation that new tools need to be forged to perform the new tasks. As little as 5 years ago, we were only thinking of tens to hundreds of gigabytes of storage for our personal computers.

Today, we are thinking in tens to hundreds of terabytes. Thus, big data is a moving target. Put another way, it is that amount of data that is just beyond our immediate grasp, e.g., we have to work hard to store it, access it, manage it, and process it. The current growth rate in the amount of data collected is staggering.

A major challenge for IT researchers and practitioners is that this growth rate is fast exceeding our ability to both: design appropriate systems to handle the data effectively and and analyze it to extract relevant meaning for decision making. In this paper we identify critical issues associated with data storage, management, and processing. The rest of the paper consists of Characteristics of Big data, Hadoop Distributed File System and Configuration Tuning Parameters.

II. BIG DATA CHARACTERISTICS

Big Data has three dimensions, volume, variety, and velocity. Big data technologies describe a new generation of technologies and architectures designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis. There are currently a number of issues and challenges in addressing these characteristics going forward. We suggest there are three fundamental issue areas that need to be addressed in dealing with big data: storage issues, management issues, and processing issues. Each of these represents a large set of technical research problems in its own right.

Data Volume

Data volume measures the amount of data available to an organization, which does not necessarily have to own all of it as long as it can access it. As data volume increases, the value of different data records will decrease in proportion to age, type, richness, and quantity among other factors.

Data Velocity

Data velocity measures the speed of data creation, streaming, and aggregation. eCommerce has rapidly increased the speed and richness of data used for different business transactions (for example, web-site clicks). Data velocity management is much more than a bandwidth issue; it is also an ingest issue.

Data Variety

Data variety is a measure of the richness of the data representation – text, images video, audio, etc. From an analytic perspective, it is probably the biggest obstacle to effectively using large volumes of data. Incompatible data formats, non-aligned data structures, and inconsistent data semantics represents significant challenges that can lead to analytic sprawl.

Data Value

Data value measures the usefulness of data in making decisions. It has been noted that "the purpose of computing is insight, not numbers". Data science is exploratory and useful in getting to know the data, but "analytic science" encompasses the predictive power of big data.

Complexity

Complexity measures the degree of interconnectedness (possibly very large) and interdependence in big data structures such that a small change (or combination of small changes) in one or a few elements can yield very large changes or a small change that ripple across or cascade through the system and substantially affect its behavior, or no change at all.

III. MAPREDUCE AND HADOOP

In the programming model of MapRedue, the input of the computation is a set of key/value pairs, and the output is also a set of key/value pairs usually in a different domain from the input. Users define a map function which converts one input key/value pair to an arbitrary number of intermediate key/value pairs, and a reduce function which merges all intermediate values of the same intermediate key into a smaller set of values, typically one value for each intermediate key. An example of the application of the programming model is counting the number of occurrences of each word in a large collection of documents. The input<key/value>pair to the map function is<the name of certain document in the collection/contents of that document>. The map function emits an intermediate key/value pair of<word/1>for each word in the document. Then the reduce function sums all counts emitted for a particular word to obtain the total number of occurrences of that word.

Hadoop is currently the most mature, accessible, and popular implementation of the MapReduce programming model. A Hadoop cluster adopts the master–slave architecture, where the master node is called the JobTracker, and the multiple slave nodes TaskTrackers. Hadoop is usually supported by the Hadoop Distributed File System (HDFS), an open-source implementation of the Google File System (GFS). HDFS also adopts the master–slave architecture, where the NameNode (master) maintains the file namespace and directs client applications to the DataNodes (slaves) that actually store the data blocks. HDFS stores separate copies (three copies by default) of each data block for both fault tolerance and performance improvement. In a large Hadoop cluster, each slave node serves as both the TaskTracker and the DataNode, and there would usually be two dedicated master nodes serving as the JobTracker and the NameNode respectively.

In the case of small clusters, there may be only one dedicated master node that serves as both the JobTracker and the NameNode. The fig.1 shows how Write operation are performed on HDFS and fig. 2 shows how Read operations are performed.

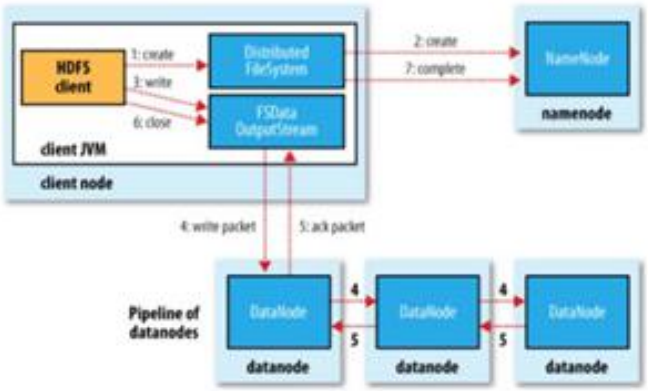
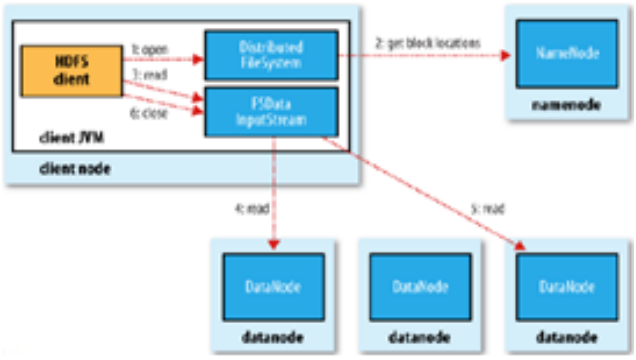


Fig. 1& 2 Hadoop Read, Write Operation



When launching a MapReduce job, Hadoop first splits the input file into fixed-sized data blocks (64 MB by default) that are then stored in HDFS. The MapReduce job is divided into certain number of map and reduce tasks that can be run on slave nodes in parallel. Each map task processes one data block of the input file, and outputs intermediate key/value pairs generated by the user defined map function. The output of a map task is first written to a memory buffer, and then written to a spill file on local disk when the data in the buffer reaches certain threshold. All the spill files generated by one map task are eventually merged into one single partitioned and sorted intermediate file on the local disk of the map task.

Each partition in this intermediate file is to be processed by one different reduce task, and is copied by the reduce task as soon as the partition becomes available. Running in parallel, reduce tasks then apply the user defined reduce function to the intermediate key/value pairs associated with each intermediate key, and generate the final output of the MapReduce job. In a Hadoop cluster, the JobTracker is the job submission node where a client application submits the MapReduce job to be executed. The JobTracker organizes the whole execution process of the MapReduce job, and coordinates the running of all map and reduce tasks. TaskTrakers are the worker nodes which actually perform all the map and reduce tasks. Each TaskTraker has a configurable number of task slots for task assignment (two slots for map tasks and two for reduce tasks by default), so that the resources of a TaskTraker node can be fully utilized.

The JobTracker is responsible for both job scheduling, i.e. how to schedule concurrent jobs from multiple users, and task assignment, i.e. how to assign tasks to all TaskTrackers. In this paper, we only address the problem of map task assignment. The map task assignment scheme of Hadoop dopts a heartbeat protocol. Each TaskTraker sends a heartbeat message to the JobTracker every few minutes to inform the latter that it's functioning properly and also whether it has an empty task slot. If a TaskTracker has an empty slot, the acknowledgment message from the JobTracker would contain information on the assignment of a new input data block. To reduce the overhead of data transfer across network, the JobTracker attempts to enforce data locality when it performs task assignment.

When a TaskTraker is available for task assignment, the JobTracker would first attempt to find an unprocessed data block that is located on the local disk of the TaskTracker. If it cannot find a local data block, the JobTracker would then attempt to find a data block that is located on certain node that is on the same rack as the TaskTracker. If it still cannot find a rack-local block, the JobTracker would finally find an unprocessed block that is as close to the TaskTracker as possible based on the topology information on the cluster.

While map tasks have only one stage, reduce tasks consist of three: copy, sort and reduce stages. In the copy stage, reduce tasks copy the intermediate data produced by map tasks. Each reduce task is usually responsible for processing the intermediate data associated with many intermediate keys. Therefore, in the sort stage, reduce tasks need to sort all the intermediate data copied by the intermediate keys. In the reduce stage, reduce tasks apply the user defined reduce function to the

intermediate data associated with each intermediate key, and store the output in final output files. The output files are kept in the HDFS, and each reduce task generates exactly one output file. The following

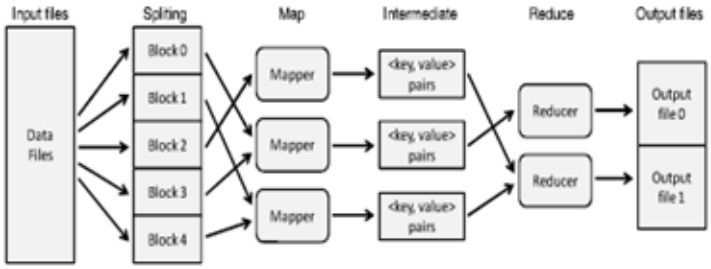


Fig.3 shows the Hadoop Map Reduce model

IV. CONFIGURATION TUNING PARAMETERS

MapReduce makes the guarantee that the input to every reducer is sorted by key. The process by which the system performs the sort—and transfers the map outputs to the reducers as nputs is known as the *shuffle.* The general principle is to give the shuffle as much memory as possible. However, there is a trade-off, in that you need to make sure that your map and reduce functions get enough memory to operate.

This is why it is best to write your map and reduce functions to use as little memory as possible—certainly they should not use an unbounded amount of memory (by avoiding accumulating values in a map, for example).
The amount of memory given to the JVMs in which the map and reduce tasks run is set by the mapred.child.java.opts property.

On the map side, the best performance can be obtained by avoiding multiple spills to disk(fig. 4); one is optimal. If you can estimate the size of your map outputs, then you can set the io.sort.* properties appropriately to minimize the number of spills. In particular, you should increase io.sort.mb if you can.

There is a MapReduce counter ("Spilled records"; that counts the total number of records that were spilled to disk over the course of a job, which can be useful for tuning. Note that the counter includes both map and reduce side spills.

Property name	Type	Default value
io.sort.mb	int	100
io.sort.record.percent	float	0.05
io.sort.spill.percent	float	0.80
io.sort.factor	int	10
min.num.spills.for. combine	int	3
mapred.compress.map. output	boolean	false
mapred.map.output. compression.codec	Class name	org.apache.hadoop.io. compress.DefaultCodec
task tracker.http.threads	int	40

Fig. 4. Mapside Tuning Parameters

On the reduce side, the best performance is obtained when the intermediate data can reside entirely in memory. By default, this does not happen, since for the general case all the memory is reserved for the reduce function. But if your reduce function has light memory requirements, then setting mapred.inmem.merge.threshold to 0(fig.5) and mapred.job.reduce.input.buffer.percent to 1.0 may bring a performance boost.

More generally, Hadoop uses a buffer size of 4 KB by default, which is low, so you should increase this across the cluster (by setting io.file.buffer.size,

Property name	Type	Default value
mapred.reduce.parallel. copies	int	5
mapred.reduce.copy.backoff	int	300
io.sort.factor	int	10
mapred.job.shuffle.input. buffer.percent	float	0.70
mapred.job.shuffle.merge. percent	float	0.66
mapred.inmem.merge.threshold	int	1000
mapred.job.reduce.input. buffer.percent	float	0.0

Fig. 5. Reducer side tuning Parameters

V. CONCLUSION

Map Reduce programming model for Hadoop improves the handling of huge amount of data effectively during Read and Write operations on HDFS which consists of actual data. The Operations like job submission, Job initialization, Job Execution an Task executions are improved significantly by the effective use of various HDFS configuration tuning parameters at Mapper and Reducer side.

REFERENCES

[1] Hadoop, http://hadoop.apache.org/.
[2] Hadoop Distributed File System, http://hadoop.apache.org/docs/stable/hdfs_design.html.
[3] Hadoop MapReduce, ttp://hadoop.apache.org/docs/stable/mapred_tutorialtml.
[4] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: OSDI '04, Dec. 2004, pp.137–150.
[5] B. He, W. Fang, Q. Luo, N. Govindaraju, T. Wang, Mars: a MapReduce framework on graphics processors, in: ACM 2008, 2008, pp.260–269.
[6] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, I. Stoica, Improving MapReduce performance in heterogeneous environments, in: Proc. OSDI, San Diego, CA, De-cember 2008, pp.29–42.
[7] Fox, B. 2011. "Leveraging Big Data for Big Impact", Health Management Technology
[8] Jacobs, A. 2009. "Pathologies of Big Data", *Communications of the ACM*, 52(8):36-44
[9] Stonebraker, M. and J. Hong. 2012. "Researchers' Big Data Crisis; Understanding Design and Functionality", Communications of the ACM, 55(2

