International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 5, May 2015 (ISSN: 2394 – 6598)

SNAPPY COMPRESSION DECOMPRESSION USING MAPREDUCE TECHNIQUE

1st Author
Hema Jadhav
KMIT, MTech CSE
hemasjadhav@gmail.com

ABSTRACT
Snappy is a compression/decompression library. It does not aim for maximum compression, or compatablity with any other compression library instead it aims for very high speeds and reasonable compression. For instance, compared to fastest mode of zlib, Snappy is an order of magnitude faster for most inputs, but the resulting compressed files are anywhere from 20% to 100% bigger. On a single core of a Corei7 processor in 64bit mode,Snappy compress at about 250 MB/secor more and decompress at about 500 MB/sec or more.

Snappy is widely used inside Google, in everything from BigTable and MapReduce to our internal RPC systems.

1. INTRODUCTION
Snappy is a compression/decompression library. It does not aim for maximum compression, or compatablity with any other compression library instead it aims for very high speeds and reasonable compression. For instance, compared to fastest mode of zlib, Snappy is an order of magnitude faster for most inputs, but the resulting compressed files are anywhere from 20% to 100% bigger. On a single core of a Corei7 processor in 64bit mode,Snappy compress at about 250 MB/secor more and decompress at about 500 MB/sec or more.

For the computer industry, effective and prompt analysis of gathered data means better understanding of consumer needs and more business. As the requirement of high storage capacity and data intensive computing grows, the scale of storage clusters increases. The key aspect of making such storage clusters cost effective and efficient is utilizing an appropriate software framework and platform for large scale computing.

2. Introduction to Hadoop
Hadoop was chosen for this thes is for several reasons. First, it is popular and widely used by a number of leading organizations including, Amazon, Facebook, Google, Yahoo! and many others. Second, it is designed for commodity hardware, significantly lowering the cost of building the cluster. Third, Hadoop is an open source technology developed in Java, making it easier to obtain, distribute and modify whenever necessary. Its effective use, lower cost and easy access makes it a potentially stable base as a large scale storage system for future technologies. Thus, research into the architecture of Hadoop file system should help data intensive applications with their rapidly growing storage needs.

3. Introduction to HDFS
Implementation of Hadoop is carried out in two main service components of the master / slave architecture. The file system metadata is decoupled from its actual data located on an individual NameNode machine. Decoupling provides flexibility to the architecture to accommodate more DataNodes in the cluster. Hadoop was chosen for this thesis for several reasons. First, it is popular and widely used by a number of leading organizations including, Amazon, Facebook, Google, Yahoo! and many others. Second, it isdesigned for commodity hardware, significantly lowering the cost of building the cluster. Third, Hadoop is an open source technology developed in Java, making it easier to obtain, distribute and modify whenever necessary. Its effective use, lower cost and easy access makes it a potentially stable base as a large scale storage system for future technologies. Thus, research into the architecture of Hadoop file system should help data intensive applications with their rapidly growing storage needs.

3.1 Data Node
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. Since HDFS is built on commodity

hardware, the machine failure rate is high. In order to make the system fault tolerant, data blocks are replicated across multiple Data Nodes. HDFS provides replication, fault detection and automatic data block recovery to maintain seamless storage access. By default replication takes place on three nodes across the cluster. When a client tries to access the failed Data Node, the Name Node maps the block replica and returns it to the client. For achieving the high throughput, the file system nodes are connected by high bandwidth network.

.

3.2 Name Node

The NameNode maintains the file system metadata as the HDFS directory tree and operates as a centralized service in the cluster. It controls the mapping between file name, data block locations and the DataNodes on which data blocks are stored. It also writes the transaction logs to record modifications in the file system. Clients communicate with the NameNode for common file system operations such as open, close, rename and delete. The namespace is a live record of the HDFS located on the centralized NameNode server. It is a directory treestructure of the file system which documents various aspects of the HDFS such as block locations, replication factor, load balancing, client access rights and file information. The namespace serves as a mapping for data location and helps HDFS clients to perform file system operations.

The metadata is stored as a file system image (fsimage) file which is a persistent checkpoint of the file system. The edit log records the write operations submitted by the file system clients. When the edit log size exceeds a predefined threshold, the NameNode moves the transactions into live memory and apply each operation to the fsimage.

A backup of the namespace is periodically stored on the local disk of the NameNode and synchronized with a secondary master node as a provision against NameNode failure.

3.3 Job Tracker

The Job Tracker is the service within Hadoop that farms out Map Reducetasks to specific nodes in the cluster, ideally the nodes that have the data, or at least are in the same rack.

1. Client applications submit jobs to the Job tracker.
2. The Job Tracker talks to the Name Nodeto determine the location of the data. The Job Tracker submits the work to the chosen TaskTracke rnodes.
3. The TaskTracke rnodes are monitored. If they do not submit heartbeat signals often enough, they are deemed to have failed and the work is scheduled on a different Task Tracker.
4. A Task Trackerwill notify the Job Tracker when a task fails. The Job Tracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may even blacklist the Task Trackeras unreliable.
5. When the work is completed, the Job Tracker updates its status.

Client applications can poll the Job Tracker for information.
The Job Tracker is a point of failure for the Hadoop Map Reduceservice. If it goes down, all running jobs are halted.

3.4 Task Tracker

A TaskTracker is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from aJobTracker.

Every Task Tracker is configured with a set of slots; these indicate the number of tasks that it can accept. When the JobT rackertries to find somewhere to schedule a task within the Map Reduceoperations, it

first looks for an empty slot on the same server that hosts the Data Nodecontaining the data, and if not, it looks for an empty slot on a machine in the same rack. The TaskTracker spawns a separate JVM processes to do the actual work; this is to ensure that process failure does not take down the task tracker. The TaskTracker monitors these spawned processes, capturing the output and exit codes. When the process finishes, successfully or not, the tracker notifies theJobTracker. TheTaskTrackersalso send out heartbeat messages to the JobTracker, usually every few minutes, to reassure the Job Trackerthat it is still alive. These messages also inform the Job Tracker. The JobTracker will first determine the number of splits (each split is configurable, ~1664MB) from the input path, and select some TaskTracker based on their network proximityto the data sources, then the JobTracker send the task requests to those selected TaskTrackers.
Each TaskTracker will start the map phase processing by extracting the input data from the splits. For each record parsed by the "Input Format", it invoke the user provided "map" function, which emits a number of key/value pair in the memory buffer. A periodic wakeup process will sort the memory buffer into different reducer node by invoke the "combine" function. The key/value pairs are sorted into one of the R local files (suppose there are R reducer nodes). When the map task completes (all splits are done), the Task Tracker will notify the Job Tracker. When all the Task Trackers are done, the JobTracker will notify the selected Task Trackers for the reduce phase.

Each TaskTracker will read the region files remotely. It sorts the key/value pairs and for each key, it invokes the "reduce" function, which collects the key/aggregatedValue into the output file (one per reducer node).

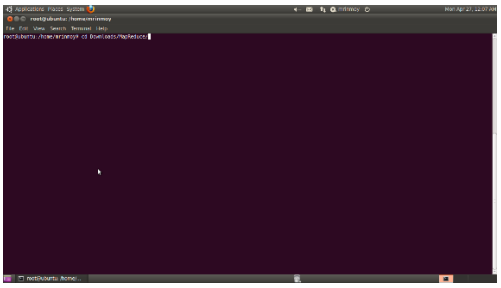
Map/Reduce framework is resilient to crash of any components. The JobTracker keep tracks of the progress of each phases and periodically ping the TaskTracker for their health status. When any of the map phase TaskTracker crashes, the JobTracker will reassign the map task to a different TaskTracker node, which will rerun all the assigned splits. If the reduce phase TaskTracker crashes, the JobTracker will rerun the reduce at a different TaskTracker.

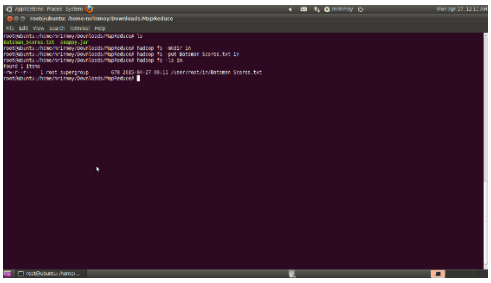
After both phases completes, the JobTracker will unblock the client program.

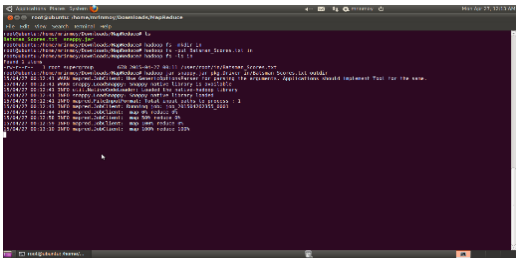
4.Proposed Model

In the proposed system we does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression. For instance, compared to the fastest mode of zlib, Snappy is an order of magnitude faster for most inputs, but the resulting compressed files are anywhere from 20% to 100% bigger. On a single core of a Core i7 processor in 64-bit mode, Snappy compresses at about 250 MB/sec or more and decompresses at about 500 MB/sec or more.

5.Screenshots







6. REFERENCES

[1] Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. .

[2] Ding, W. and Marchionini, G. 1997 A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[3] Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems

[4] Tavel, P. 2007 Modeling and Simulation Design. AK Peters Ltd.Sannella, M. J. 1994 CSatisfaction and Debugging for Interactive User Interfaces. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.

[5] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3 (Mar. 2003), 1289-1305.

[6] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE.

[7] Y.T. Yu, M.F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions", Journal of Systems and Software, 2005, in press.

[8] Spector, A. Z. 1989. Achieving application requirements. In Distributed Systems, S. Mullender