International Journal of scientific research and management (IJSRM) ||Volume 2 ||Issue 11||Pages|| 1731-1735 ||Nov 2014||
Website: www.ijsrm.in ISSN (e): 2321-3418

A Paradigm Shift for Improved Processing of Small Files in Hadoop

Miss Priyanka G. Phakade[1], Dr. Suhas D. Raut[2]

[1]N.K. Orchid College of Engineering and Technology, Solapur
priyankaphakade@gmail.com

[2]N.K. Orchid College of Engineering and Technology, Solapur
suhasraut@gmail.com

Abstract: *HDFS is designed to handle large files containing petabytes or exabytes of data. However, there are plenty of applications that need access & manipulation of large number of small files. HDFS suffers performance penalty while dealing with large number of small files. With the rapid development of Internet, users may tend to store their data and programs in a cloud computing platform. Personal data has an obvious feature –large number and small file size. In such cases, HDFS struggles to meet performance criteria. In hadoop architecture, the FileInputFormat generates a split per file. Map tasks usually process a block of input at a time. In case of large number of small files, each map task processes very little input. Every map task imposes extra bookkeeping overhead on NameNode and also consumes considerable time to process large number of small files. We have attempted a strategy wherein large number of small files are clubbed together to form a single split. This inherently reduces number of blocks generated by FileInputFormat, resulting in lesser processing time. Clubbing of small files is achieved through a customized mapper. Practical setup has indicated performance improvement of around 80%. The paper covers this paradigm shift in processing large number of small files in HDFS for performance improvement.*

Keywords: Hadoop Distributed File System; mapreduce; map task; small files**.**

1. Introduction

The Hadoop Distributed File System (HDFS) [7] is a distributed file system designed to run on commodity hardware. HDFS provides efficient access to application data and is suitable for applications having big data sets[6]. Hadoop is an open-source software framework developed for reliable, scalable, distributed computing and storage [19]. It is gaining increasing popularity for building big data ecosystem such as Cloud based on it [14]. Hadoop distributed file system (HDFS), which is inspired by Google file system [13], is composed of HDFS is a representative for Internet service file systems running on clusters [15], and is widely adopted to support lots of Internet applications as file systems. HDFS is designed for storing large files with streaming data access patterns [16], and stores small files inefficiently. The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts [17].

2. Small File Problem

The Hadoop Distributed File System (HDFS) is a distributed file system. It is mainly designed for batch processing of large volume of data. The default block size of HDFS is 64MB. HDFS does not work well with lot of small files for the two reasons. First reason is that each block will hold a single file. Thus, if there are so many small files then there will have a lot of small blocks (smaller than the configured block size). Reading all these blocks, one by one means a lot of time will be spent with disk seeks. Another reason is the NameNode keeps track of each file and each block (about 150 bytes for each) and stores this data into memory. A large number of files will occupy more memory [20]. The performance of the Hadoop Distributed File System (HDFS) decreases dramatically when handling interaction-intensive files, i.e., files that have relatively small size but are accessed frequently[10].

3. Related Work

Researches on small file storage on HDFS can be classified into two categories: general solutions and special solutions to meet particular scenarios [1]. General solution includes HAR, SequenceFile. Mackeyet al [4] utilized HAR to improve the metadata management of HDFS for small files. New Hadoop Archive (NHAR) designed the architecture of HAR in order to improving performance of small-file accessing in Hadoop[3]. As for special solutions, HDWebGIS [9] is an interesting solution. Liu et al [8] proposed an approach to optimize I/O performance of Geographic data on HDFS. Jiang et al[5] proposed HDFS I/O optimization by using local cache to save some metadata of small files to reduce usage of NameNode. EHDFS is an extension of the work done by Dong et al. for handling small files in HDFS[2]. Hadoop is an open-source Apache project. Yahoo! has developed and contributed to 80% of the core of Hadoop [4] from Yahoo! described the detailed design and implementation of HDFS. They realized that their assumption that applications would mainly create large files was flawed, and new classes of applications for HDFS would need to store a large number of smaller files. As there is only one NameNode in Hadoop and it keeps all the metadata in main memory, a large number of small files produce significant impact on the metadata performance of HDFS, and it appears to be the bottleneck for handling metadata requests of massive small files [12]. HDFS is designed to read/write large files, and there is no optimization for small files. Mismatch of accessing patterns will emerge if HDFS is used to read/write a large

amount of small files directly (Liu et al., 2009). Moreover, HDFS ignores the optimization on the native storage resource, and leads to local disk access becoming a bottleneck. In addition, data prefetching is not employed to improve access performance for HDFS[11]. HAR is a general small file solution which archives small files into larger files [18].

4. Proposed System

The Hadoop Distributed File System (HDFS) is developed to store and process large data sets over the range of terabytes & petabytes. However, storing a large number of small files in HDFS is inefficient. Also too many small files increases the number of mappers, less input for each map task and overall processing time. Hadoop works better with a small number of large files than a large number of small files. One reason for this is that FileInputFormat generates splits in such a way that each split is all or part of a single file. Compare a 1 GB file broken into sixteen 64 MB blocks and 10,000 or so 100 KB files. The 10,000 files use one map each, and the job time can be tens or hundreds of times slower than the equivalent one with a single input file & it uses 16 map tasks. The situation is alleviated by CombineFileInputFormat, which was designed to work well with small files. FileInputFormat creates a split per file. CombineFileInputFormat packs many files into each split so that each mapper has more to process. Therefore, as the each mapper gets sufficient input to process, the job completion time becomes less. In this way, this method minimizes the overall processing time.

Fig.1 shows the proposed system for small files. In first step, we have implemented the middleware between the HDFS & HDFS Client. HDFS Client requests for connection to NameNode. NameNode accepts the connection. This is implemented by socket. HDFSClient wish to store files in HDFS. For this purpose HDFS client want to take permission from NameNode. NameNode can grant or deny the permission. This is achieved by using DataOutputFormat. If NameNode does not allow to HDFS Client for storing files into HDFS then HDFS Client cannot store the files into HDFS and the connection between the HDFS Client & NameNode is closed. Otherwise HDFS Client can store the files into HDFS. Once the small files are stored into HDFS then NameNode starts to process these files. The maximum split size is defined as 64MB. The JobTracker which resides on NameNode will pack many files into one split (until split size becomes 64MB) by using CombineFileInputFormat. Splits are provided as an input to each DataNode. The TaskTracker which resides on DataNode will processes the input using map & reduce tasks. The input for map tasks is key-value pair. With each line a key provided for mapper consists of the file name and the offset length of that line & Text is a value. When a map task is completed it will generates the intermediate result which is given to reducer. After getting the input, the reducer gives sort-merge output & finally the result is stored on HDFS. Here, multiple reducers are used for taking advantage of parallelism.

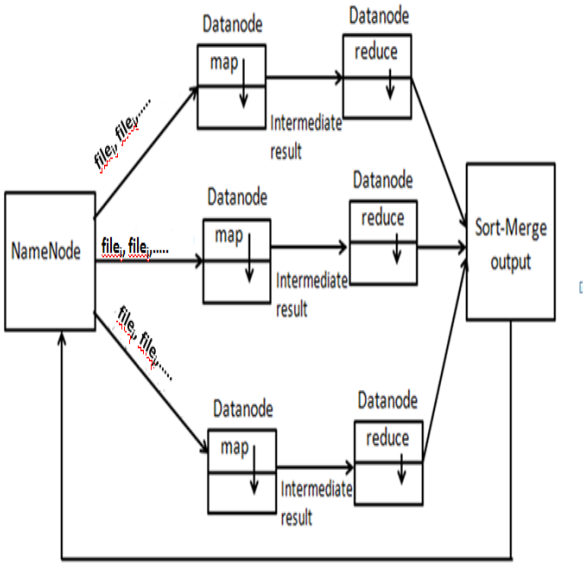


Figure 1. Workflow of Prposed System

The proposed system is differentiated than the normal hadoop architecture in such a way that the input provided for map tasks are more. As single reducer will surely bring down performance due to lack of parallelism the multiple reducers are used.

5. Evaluation and Results

5.1Experimental Environment

The test platform contains a cluster comprising 6 machines (1 NameNode, 4DataNodes & 1Client).

Each of these machines has the following configuration:

1) Intel(R) Core(TM)2 Duo CPU T6570 @ 2.10GHz

2) 2GB RAM

3) 160 GB SATA HDD

All the machines are connected using 1 GBPS Ethernet network. In each machine, Ubuntu 13.04 with the kernel version of 3.8.0-19.29-generic 3.8.8 is installed. Hadoop version 1.2.1 and Java version open-jdk-7 have been used. The number of replicas for data blocks is set to 6 and the default block size is 64MB.

The number of mapper used is one & multiple reducers are used. The number of reducers has been set to 13. We compare our system with word-count application of Hadoop. Word-count counts the number of unique words in large input text files. Finally, we have calculated the processing time for word-count application as well as our proposed system.

5.2 Workload Overview

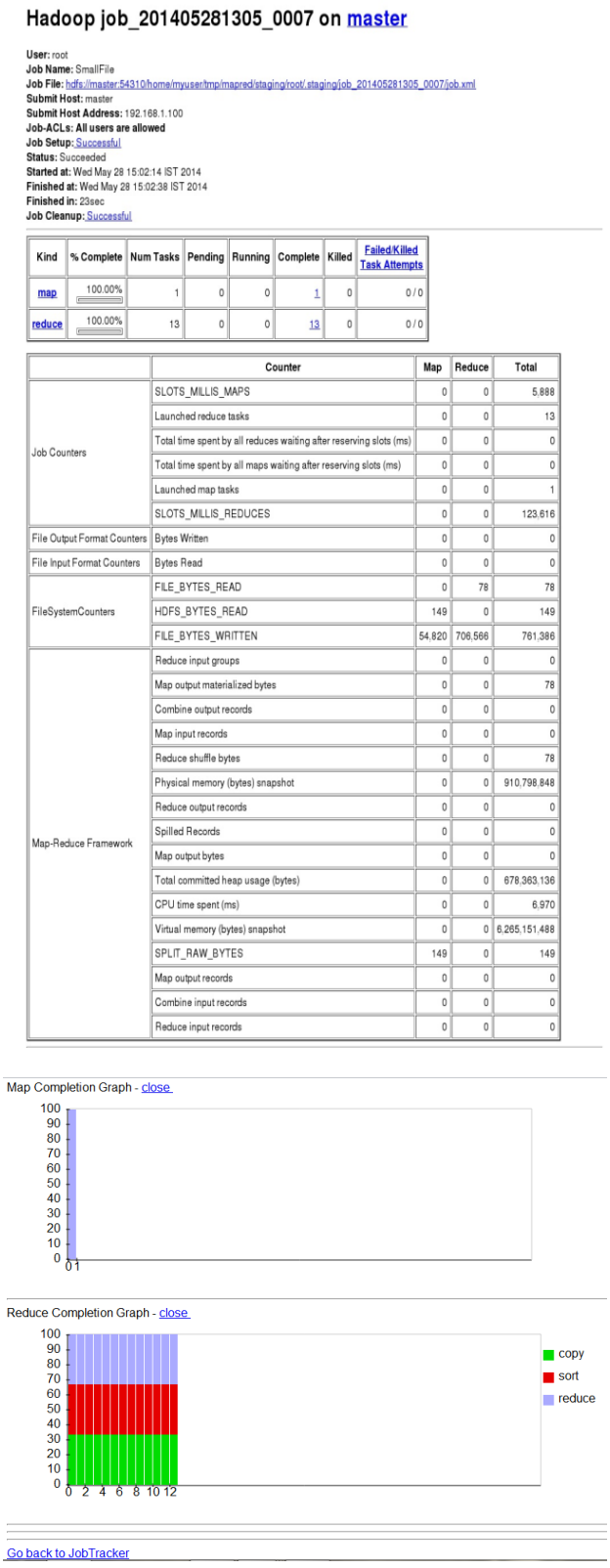
In existing system, the Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Reducer reduces a set of intermediate values which share a key to a smaller set of values. Multiple mappers & Single reducer is used for Word-Count application. On the other hand, in proposed system the MapReduce framework spawns one map task for each split generated by the CombineFileInputFormat. So, the number of files are grouped into one split. Each split is processed by map task. The map task gives the intermediate output to multiple reducers so that the result brought out in much less time. As shown in result, hadoop job is executed on NameNode. The files used for processing are 15,000 in number. HDFS Client gives input of 5000 files to HDFS System. The NameNode grants the permission to store the files into the HDFS System. After getting permission from NameNode, the files are stored in HDFS System. Then One map task & thirteen reduce tasks are assigned to process those 5000 files. The time required to process those files is 23 seconds. On the other hand, in existing system, the word count application assigns 2546 map tasks & one reduce task to process those files. Processing time for those files is 318 seconds. The workload for the processing time measurement contains a total of 15,000 files. The size of these files range from 10KB to 150KB. Following table shows time required to process the small files in seconds.

TABLE I

PROCESSING TIME REQUIRED FOR RESPECTIVE SYSTEM

Number of Files	Processing time for Proposed System (sec)	Processing time for WordCount (sec)
1000	26	318
2000	25	605
3000	23	878
4000	24	1179
5000	23	1697

.



Hadoop job_201405281305_0007 on master

User: root
Job Name: SmallFile
Job File: hdfs://master:54310/home/myuser/tmp/mapred/staging/root/.staging/job_201405281305_0007/job.xml
Submit Host: master
Submit Host Address: 192.168.1.100
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Wed May 28 15:02:14 IST 2014
Finished at: Wed May 28 15:02:38 IST 2014
Finished in: 23sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	1	0	0	1	0	0 / 0
reduce	100.00%	13	0	0	13	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	SLOTS_MILLIS_MAPS	0	0	5,888
	Launched reduce tasks	0	0	13
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
	Total time spent by all maps waiting after reserving slots (ms)	0	0	0
	Launched map tasks	0	0	1
	SLOTS_MILLIS_REDUCES	0	0	123,616
File Output Format Counters	Bytes Written	0	0	0
File Input Format Counters	Bytes Read	0	0	0
FileSystemCounters	FILE_BYTES_READ	0	78	78
	HDFS_BYTES_READ	149	0	149
	FILE_BYTES_WRITTEN	54,820	706,566	761,386
Map-Reduce Framework	Reduce input groups	0	0	0
	Map output materialized bytes	0	0	78
	Combine output records	0	0	0
	Map input records	0	0	0
	Reduce shuffle bytes	0	0	78
	Physical memory (bytes) snapshot	0	0	910,798,848
	Reduce output records	0	0	0
	Spilled Records	0	0	0
	Map output bytes	0	0	0
	Total committed heap usage (bytes)	0	0	676,363,136
	CPU time spent (ms)	0	0	6,970
	Virtual memory (bytes) snapshot	0	0	6,265,151,488
	SPLIT_RAW_BYTES	149	0	149
	Map output records	0	0	0
	Combine input records	0	0	0
	Reduce input records	0	0	0

Map Completion Graph - close

Reduce Completion Graph - close

Go back to JobTracker

The following result shows that the proposed system requires 23 sec for processing 5000 small files of 15KB to 150KB size.

The following result shows that the Word Count application requires 28 min, 17 sec for processing 5000 small files of 15KB to 150KB size.

Hadoop job_201405281305_0021 on master

User: root
Job Name: word count
Job File: hdfs://master:54310/home/myuser/tmp/mapred/staging/root/.staging/job_201405281305_0021/job.xml
Submit Host: master
Submit Host Address: 192.168.1.100
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Wed May 28 17:19:28 IST 2014
Finished at: Wed May 28 17:47:46 IST 2014
Finished in: 28mins, 17sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	2546	0	0	2546	0	0 / 4
reduce	100.00%	1	0	0	1	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	SLOTS_MILLIS_MAPS	0	0	16,204,635
	Launched reduce tasks	0	0	1
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
	Total time spent by all maps waiting after reserving slots (ms)	0	0	0
	Launched map tasks	0	0	2,550
	Data-local map tasks	0	0	2,550
	SLOTS_MILLIS_REDUCES	0	0	1,601,752
File Output Format Counters	Bytes Written	0	0	1,128,136
File Input Format Counters	Bytes Read	0	0	9,693,446,937
FileSystemCounters	FILE_BYTES_READ	4,301,993,578	97,593,783	4,399,587,361
	HDFS_BYTES_READ	9,693,761,024	0	9,693,761,024
	FILE_BYTES_WRITTEN	6,826,502,493	97,648,356	6,924,150,849
	HDFS_BYTES_WRITTEN	0	1,128,136	1,128,136
Map-Reduce Framework	Reduce input groups	0	0	82,336
	Map output materialized bytes	0	0	2,385,282,015
	Combine output records	0	0	450,941,535
	Map input records	0	0	210,136,228
	Reduce shuffle bytes	0	0	2,385,282,015
	Physical memory (bytes) snapshot	0	0	491,785,580,544
	Reduce output records	0	0	82,336
	Spilled Records	0	0	469,948,282
	Map output bytes	0	0	16,356,115,132
	Total committed heap usage (bytes)	0	0	432,397,520,896
	CPU time spent (ms)	0	0	10,638,460
	Virtual memory (bytes) snapshot	0	0	1,129,453,395,968
	SPLIT_RAW_BYTES	314,087	0	314,087
	Map output records	0	0	1,691,938,535
	Combine input records	0	0	2,137,365,664
	Reduce input records	0	0	5,514,406

Conclusion

In this paper, we introduce the improved model for processing of small files. It requires modification in the input format and task management of the mapreduce framework. HDFS is designed to store large files. It cannot process large number of small files efficiently. Small file is the one that significantly smaller than an HDFS block (64MB). If there are so many small files then a block is assigned to each file. Map task processes block of input at a time, therefore each map task processes very little input. So, there is need of providing more input to Mapper. Hence, the number of small files are combined into one split. Each split processed by single map task. In this way, the number of map tasks are reduced. Each map task processes multiple blocks at a time so that each mapper gets more input to process. Thus, the processing time required for large number of small files is minimized.

Acknowledgement

The Author place on record and warmly acknowledges the continuous encouragement, invaluable supervision, timely suggestions and inspired guidance offered by guide Dr. Suhas Raut, Professor, Department of Computer Science and Engineering, Nagesh Karajagi Orchid College of Engineering & Technology, Solapur.

References

[1] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, Y. Li. "A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files". In Proceedings of IEEE International Conference on Services Computing, Miami, Florida, USA, July 2010, pp. 65 72.

[2] Chandrasekar S, Dakshinamurthy R, Seshakumar P G, Prabavathy B, Chitra Babu "A Novel Indexing Scheme for Efficient Handling of Small Files in Hadoop Distributed File System" IEEE 2013 International Conference on Computer Communication and Informatics (ICCCI-2013), Jan. 04 - 06, 2013, Coimbatore, INDIA

[3] ChatupornVorapongkitipun, Natawut Nupairoj, "Improving Performance of Small-File Accessing in Hadoop" 2014 11th International Joint Conference on Computer Science and Software Engineering (JCSSE)

[4] G. Mackey, et al., "Improving metadata management for small files in HDFS," in Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, 2009, pp. 1-4.

[5] J. Liu, et al., "THE optimization of HDFS based on small files," in Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference on, 2010, pp. 912-915.

[6] Kala Karun. A, Chitharanjan. K "A Review on Hadoop – HDFS Infrastructure Extensions" In Proceedings of 2013 IEEE Conference on Information and Communication Technologies (ICT 2013)

[7] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The Hadoop Distributed File System," 26 th IEEE Symposium on Mass Storage Systems and technologies, Yahoo!, Sunnyvale, pp. 1-10, May 2010

[8] L. Xuhui, et al., "Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on

HDFS," in Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, 2009, pp. 1-8.

[9] Xiayu Hua ; Dept. of Comput. Sci., Illinois Inst. of Technol., Chicago, IL, USA ; Hao Wu ; Shangping Ren, "Enhancing Throughput of Hadoop Distributed File System for Interaction-Intensive Tasks" In IEEE International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2014

[10] Shafer J, Rixner S, Cox A. The hadoop distributed filesystem: balancing portability and performance. In: IEEE international symposium on performance analysis of systems & software (ISPASS). IEEE; 2010. p. 122–33.

[11] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," Proc. of the 19th ACM Symp. on Operating System Principles, pp. 29–43, 2003.

[12] Feng Wang, Jie Qiu, Jie Yang, Bo Dong, Xinhui Li, and Ying Li, "Hadoop high availability through metadata replication," Proc. of the First CIKM Workshop on Cloud Data Management, pp. 37-44, 2009.

[13] W. Tantisiriroj, S. Patil, and G. Gibson, "Data-intensive file systems for internet services: A rose by any other name," Tech. Report CMU-PDL-08-114, Oct. 2008.

[14] Tom White. Hadoop: The Definitive Guide. O'Reilly Media, Inc. June 2009.

[15] Apache hadoop http://en.wikipedia.org/wiki/Apache_Hadoop

[16] Hadoop archives, http://hadoop.apache.org/common/docs/current/hadoop_archives.html.

[17] Hadoop official site, http://hadoop.apache.org/.

[18] Solving the "Small Files Problem" in Apache Hadoop: Appending and Merging in HDFS http://pastiaro.wordpress.com/2013/06/05/solving-the-small-files-problem-in-apache-hadoop-appending-and-merging-in-hdfs/