International Journal of scientific research and management (IJSRM)
||Volume||2||Issue||12||Pages||1781-1784||2014||
Website: www.ijsrm.in ISSN (e): 2321-3418

HDFS with cache system – a paradigm for performance improvement

Ms. Archana Kakade, Dr. Suhas Raut

[1] Master of Engineering student, Department of Computer science & Engineering
NK Orchid College of Engineering & Technology, Solapur
Maharashtra, India
kakadeas34@gmail.com

[2] Ph.D. Professor, Department of Computer science & Engineering
NK Orchid College of Engineering & Technology, Solapur
Maharashtra, India
suhasraut@gmail.com

Abstract: *Due to online activities and use of resources related to computing, data is being generated at an enormous rate. To access and handle such huge amount of data spread, distributed systems is an efficient mechanism. One such mechanism is a Hadoop distributed file system (HDFS). However HDFS faces performance drawback. Hence need is felt improve upon the performance. In this paper we are presenting a new paradigm for improving small file processing in HDFS. The paradigm shift is to use cache information. It is known that accessing data from cache is much faster as compared to disk access. The cache memory is used to store frequently accessed data & hence process it much more quickly. This paper describes the system architecture that aims to provide a cache system to HDFS, we can avoid unnecessary trips HDD to fetch data and thus avoid delay.*

Keywords: Hadoop, Hadoop distributed file system (HDFS), Cache system, main Memory.

I. INTRODUCTION

Apache Hadoop [1] is well known project that includes open source implementation of a distributed file system and MapReduce. One of the significant designed features of the Hadoop system is high throughput which is extremely suitable for handling large scale data analysis and processing problems. HDFS [2] [3] is designed for write-once-read-many access model for files. In HDFS file reading may contain several interactions of connecting NameNode and DataNodes, which considerably decrease the access performance when the system is under a heavy workload.

In past decades, disk technology has evolved rapidly. Disk technology evolved into the design of complex storage systems that can host petabytes of data. Hadoop [1], MapReduce [5], Dryad [10] and HPCC (High-Performance Computing Cluster) [12] frameworks are Data-intensive and they rely on disk based file systems to meet their exponential storage demands. Hadoop distributed file system (HDFS) [6] has the capability to store huge amounts of data. There are various mechanisms to minimize disk access latencies such as jobs are scheduled on the same node that hosts the associated data, in addition, data is replicated to different nodes in numerous ways to improve throughput and job completion time.

This Paper present mechanism which stored frequently accessed data in main memory and hence processes it much more quickly. The processing speed is increased by making the data needed available in main memory. By providing a cache system to HDFS, we can avoid unnecessary trips to hard disk to fetch data and thus avoid delay.

II. RELATED WORK

Zhang et al proposed a system named as HDCache [14] which runs as a daemon on the host. HDCache is built on the top of HDFS and they are loosely coupled. This system is viewed as Client Server architecture. The third-party need to do is to integrate with a client-side dynamic library. The third-party applications use the cache to access data stored in HDFS transparently. Gurmeet Singh et al proposed another idea named as MemCached [9]. MemCached is a group of servers and data is placed into their RAM. When a particular node is in need of a data, it generates two requests. One is directed to NameNode and other one is directed to MemCached. MemCached can be used for accelerating MapReduce tasks on a Hadoop cluster. But the drawback is increased traffic overhead that will be generated from the new components MemCached that have added to Hadoop.

III. THE ARCHITECTURE

Figure 1 shows the proposed architecture [5]. HDFS's User will make a request for file. This request goes from Cache. When request is received by cache system it follows following steps:

- Cache system checks whether requested file is available in cache local memory or not.
- If requested file is available then request is fulfilled by cache. Hence here we can avoid disk access to fetch a file.

- Else client communicates with DataNodes to check whether requested file is present in their local memory. If file is available then request is fulfilled.
- Else if file is not available in cache local memory then file is fetched from disk by using HDFS API.

Consistency Validator is used to keep files present in cache local memory consistence with disk.

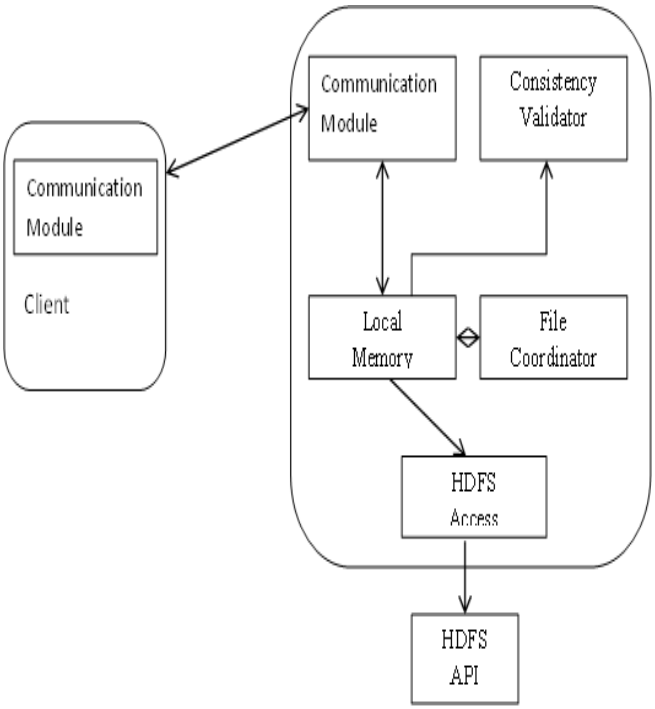


Figure 1. The Architecture

A. ***Communication Module:*** In the proposed system client communicates with HDFS in the same way as it communicates with HDFS without cache system. The communication protocols build on the TCP/IP protocol. HDFS clients connect to a Transmission Control Protocol (TCP) port opened on the NameNode, and then communicate with the name node using a proprietary Remote Procedure Call (RPC)-based protocol. Data nodes talk to the NameNode using a proprietary block-based protocol. DataNodes continuously loop, asking the NameNode for instructions. Each DataNode maintains an open server socket so that client code or other DataNodes can read or write data. The host or port for this server socket is known by the NameNode, which provides the information to interested clients or other DataNodes.

B. ***Consistency Validator:*** When a client requests for data, consistency validation is performed on request information. Consistency Validator retrieves the day of the month and time of last change of data. And it compares that information with current date and time. If the current date and time is greater than last modification time than the client is admitted to read data. Else client is not allowed to take data.

C. **File Co-ordinate Module:** This module maintains a list of files which are currently presents in local memory.

D. **Local Memory:** Local memory contains frequently requested files.

E. **HDFS Access Module:** If requested file is not present in local memory then the requested file will be fetch from disk.

IV. Implementation And Experimental Setup

The HDFSCache system is deployed on HDFS NameNode, DataNodes, and other application systems that can access HDFS through network and need cache functions. The client needs to integrate with HDFSCache in and they can access the cache services. Figure 2 shows experimental setup diagram.

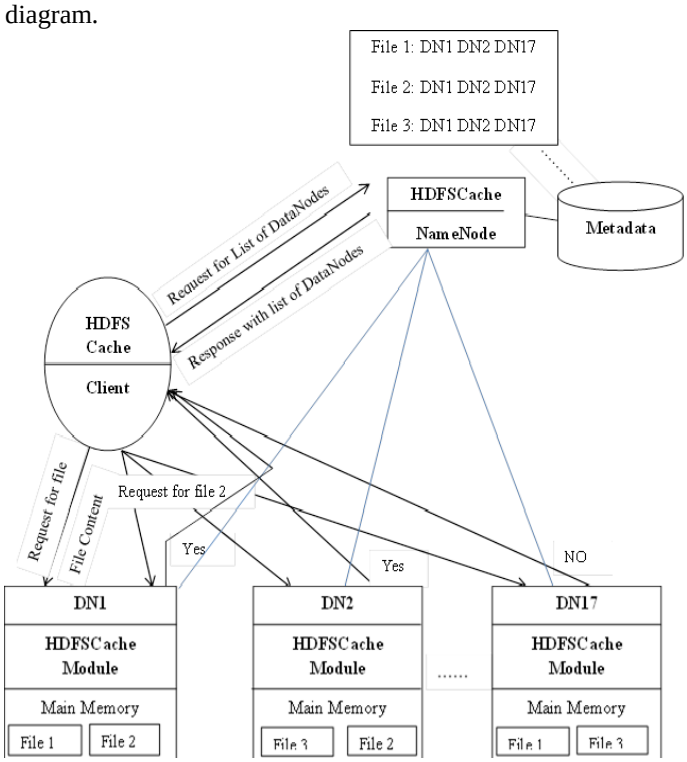


Figure 2. Experimental Setup Diagram

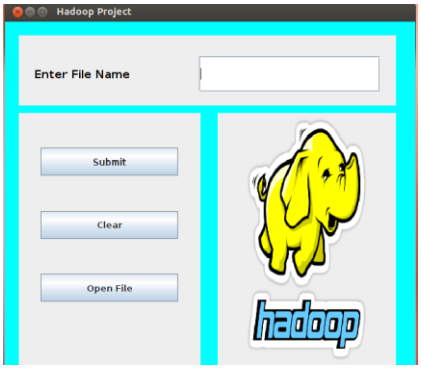
We can explain our experimental setup with one example. Consider 17 DataNodes deployed with HDFSCache Module of DataNode. Also HDFSCache is present on the Client and NameNode. Actually there are 8000 files deployed on the Hadoop Cluster. Consider 3 files out of 8000 files namely file 1 file 2 and file 3. As shown in figure these three files are replicated on DataNode 1 DataNode 2 and DataNode 17. File 1 is present in the main memory of DN 1 and DN 17. File 2 is present in the main memory of DN 1 and DN 2. And File 3 is present in the main memory of DN 2 and DN 17. When a client requests for file 2, HDFSCache Module of Client makes a request for a list of DataNodes on which that file is present to NameNode. HDFSCache Module present on NameNode response to the request with a list of DataNodes

that is the IP addresses of DataNode 1, 2 and 17 because file 2 is present on DataNodes 1, 2 and 17. The HDFSCache module of client communicates to DataNodes 1, 2 and 17.
HDFSCache Module asks these DataNodes whether file 2 is present in their main memory or not. HDFSCache module of DataNode first checks whether file 2 is present in main memory or not. If file 2 is present in main memory, then the HDFSCache Module on DataNode send a response as yes or no to the Client. In figure DN 1 and DN 2 sends a reply as a yes to the client. And DN 17 sends a reply as No to Client because DN 17 doesn't have file 2 in its main memory. Now HDFSCache Module of Client request for file content to DataNode 1. DN 1 sends file content to Client.

A. HDFSCache Module of DataNode: Each of the DataNode has their dedicated HDFSCache Module who has responsibilities of keeping track of files which are present in main memory. Another responsibility of it is communicating with HDFSCache Module of Client. Recently opened files are stored into main memory. When the request for a file is arrived which is opened is fulfilled from main memory. Hence we can avoid disk access.

B. HDFSCache Module of NameNode: The responsibility of this module is in response to client with a list of DataNodes when a client asks for it.

C. HDFSCache Module of Client: This module takes a file name from the user through the GUI. The GUI is as shown in the screenshot. As shown in the screenshot there is a text field to enter the file name. There are three buttons.



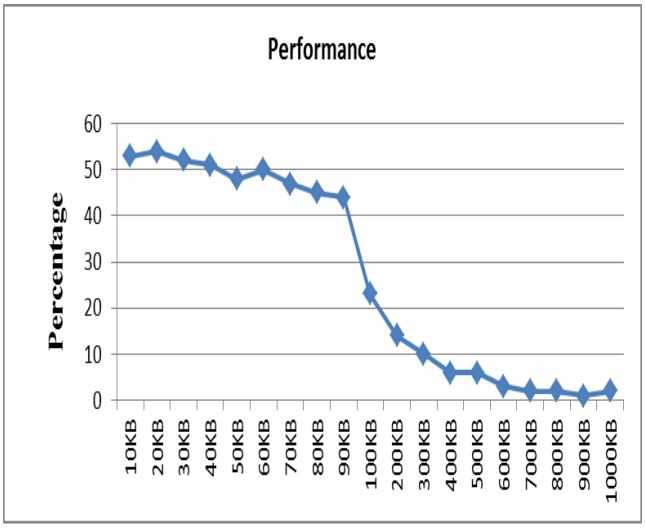
Screenshot 1 GUI

V. EVALUATION

HDFSCache system is implemented in Linux. We chose 64bit OS because 64bit OS is currently the mainstream in commercial environments with the RAM of 2 GB or more, so it can provide large memory addressing space. Taking into account the performance, we implemented the system
with Java program language. We setup a test-bed consisting of twenty five servers running Ubuntu 12.04 64bit OS. On every computer Hadoop 1.1.2 (stable version) is installed with the block size 64 MB. Twenty four of these computers are configured to be DataNode servers and the remaining one is configured to be NameNode server. On every DataNode, we deploy HDFSCache system. HDFSCache system is also deployed on both Client and NameNode. We have deployed 8000 files on the cluster. Files may be docx, text, video, audio, java file, xml file, pptx file. Files size varies from 10 KB to 64 MB. The dataset prepared consisted of data files in the increments of 10 from 10KB to 100KB and in increments of 100 from 100KB to 1000KB. The dataset consist of text files, doc files, videos, images. The data for the files was taken from the College. Table 1 shows the time required to complete the read operation by HDFS and HDFSCache. From table it is seen that for the file size of 10 KB, time required by HDFS is 7.6092 milliseconds and that for HDFSCache is 0.0597 which shows the performance increase of 53%. Hence the difference of time observed is 7.5494 milliseconds. Similar, performance increase is shown for the other files size ranging from 10 KB to 1000KB. Graph 1 shows the graph of percentage against the file size. Performance shows result of reading efficiency. The performance declines with the increase of file size. However, the size of the file exceeds 1MB, the performance decreases. As per the graph it is seen that the performance increases for the file size ranging from 10KB to 1000KB from 53% to 2%. But once the file size exceeds 1000KB, the graph shows the decrease in the performance. Which in turn result that HDFSCache performance is good for the file size upto 1000KB.

Table 1 Time required executing job (In milliseconds)

File Size	HDFS	HDFSCache
10KB	7.609245	0.059748
20KB	6.605106	0.045906
30KB	7.550665	0.069226
40KB	7.623110	0.099467
50KB	6.663618	0.149039
60KB	7.598473	0.103903
70KB	6.622031	0.169268
80KB	6.653922	0.215506
90KB	6.622009	0.228031
100KB	6.989729	0.281367
200KB	6.699742	0.471439
300KB	7.688346	0.740193
400KB	6.644013	0.980522
500KB	6.665329	0.923059
600KB	6.675682	1.869722
700KB	7.611090	3.721735
800KB	6.709381	1.982028
900KB	6.678156	3.404119
1000KB	6.867022	2.256146



Graph 1 Performance of job execution Line graph

VI. CONCLUSION

There are more and more real-time services coming forth on the internet in the big data area. Industrial circles tend to use mature technologies such as Hadoop. To improve the performance of the Hadoop distributed file system, this paper describes a novel cache system built on HDFS named HDFSCache. HDFSCache uses the main memory of DataNodes where currently accessed files are stored. Hence the request for the file is fulfilled from the main memory of DataNode. In this way we can avoid disk access. This improvement makes most of the file operations complete in the main memory, which brings down the frequency of access to disk.

Presently, this work is still first step. HDFSCache shows the performance increase for small files upto 53%. However, there decrease in performance once the file exceeds 1MB. Hence the future scope is to improve the cache system to support files more than 1000KB with multiple NameNode. In this paper cache coherence is not taken into account. So in future cache coherence is used to improve the cache system.

REFERENCE

[1] Apache Hadoop. Available at Hadoop Apache.

[2] Apache Hadoop Distributed File System. Available at Hadoop Distributed File System Apache.

[3] Scalability of Hadoop Distributed File System.

[4] D. Borthakur (2011) et al. "Apache Hadoop goes real-time at Facebook", In Proceedings of the 2011 International Conference on Management of Data (SIGMOD'11), New York, 2011.

[5] Hadoop Distributed File System with Cache technology by Archana Kakade and Dr. Suhas Raut, Industrial Science Vol.1,Issue.6/Aug. 2014 **ISSN** : 2347-5420

[6] J. Dean and S. Ghemawat (2004), "Mapreduce: Simplified Data Processing on Large Clusters". In Proceeding of the 6th Conference on Symposium on operating Systems Design and Implementation (OSDI'04), Berkeley, CA, USA, 2004, pp.137-150.

[7] D. Borthakur. The Hadoop Distributed File System: Architecture and Design. The Apache Software Foundation, 2007.

[8] Gurmeet Singh,Puneet Chandra, Rashid Tahir "A Dynamic Caching Mechanism for Hadoop using Memcached".

[9] S. Ghemawat, H. Gobioff and S. T. Leung (2003), "Google File System", In Proc. of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP'03), Lake George New York, 2003, pp.29-43.

[10] "The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM" Department of Computer Science Stanford University.

[11] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In EuroSys'07, pages 59–72, 2007.

[12] J. Shafer and S Rixner (2010), "The Hadoop distributed file system: balancing portability and performance", In 2010 IEEE International Symposium on Performance Analysis of System and Software (ISPASS2010), White Plains, NY, March 2010. Pp.122-133.

[13] HPCC Wikipedia HPCC.

[14] Jing Zhang, Gongqing Wu, Xuegang Hu, Xindong Wu (2012), "A Distributed Cache for Hadoop Distributed File System in Real-time Cloud Services". In 2012 ACM/IEEE 13th International Conference on Grid Computing.

[15] S. Zhang, J. Han, Z. Liu, K. Wang (2009), "Accelerating MapReduce with Distributed Memory Cache", In 15th International Conference on Parallel and Distributed Systems (ICPADS09), Shenzhen, 2009, pp.472-478.