A METHOD OF ADJUSTING THE NUMBER OF REPLICA DYNAMICALLY IN HDFS

Bing Li and Ke Xu
School of Computer Science and Technology, Beijing University of Posts and Telecommunications, Beijing, China

Keywords: Hadoop, HDFS, Network congestion, Hot spots detection, Adaptive replica control.

Abstract: With the development of Cloud Computing, Hadoop –an Infrastructure as a Service open source project of Apache – has been used more and more widely. As the basis of Hadoop, the Hadoop Distributed File System (HDFS) provides basic function of file storage. HDFS-an open source implementation of Google File System (GFS) –was designed for specific demand. Once the demand was changed, HDFS cannot fit it very well. Especially when the access demand of file is different, there will be hot spots and the existing replica will not be enough. It will lower the efficiency of the whole system. This paper introduced a system-level strategy which could adjust the replica number of specific file dynamically. And the experiment shows that this mechanism can prevent the problem of decline of user experience bring by hot spots and improve the overall efficiency.

1 INTRODUCTION

In the era of information explosion, there are billions of Internet services provided to users and increasing sharply every day. As a result, user's interest is changing every minute and hard to be traced. So developers can hardly know whether their service could attract enough users to cover their expenses. Nowadays Cloud Computing seems to be one of the solutions to this problem. Cloud Computing could provide limitless ability of storage and computing according to developer's demand. If the service was welcomed and have thousands of users, developer can order extra resources from Cloud Computing provider. If users were not interested any more, the developer can rent less resource to reduce their cost. In a word, developers will not necessary to buy a lot of hardware by the estimation of market size. That's why Cloud Computing costs much less than the traditional ways.

Therefore, Cloud Computing becomes a hot topic in industry area and academy area now. A lot of organizations and institutes have been developed their own Cloud Computing framework. There are also a lot of open source Cloud Computing projects and Hadoop (Apache Hadoop, 2011) which is sponsored by Apache foundation is one of them. Hadoop is an open source framework that implements the MapReduce parallel programming model (J. Dean and S. Ghemawat, 2004). Yahoo! and Facebook contribute a lot to this project and use Hadoop in their real computing environment.

There are lots of components in Hadoop such as Hadoop Common, HDFS, and MapReduce.

In this paper, section II gives a brief introduction to HDFS, especially about the flaws and the solutions; section III talks about auto hot spot detector in HDFS, section IV introduces the adaptive replica controller, section V shows the experiment data and section VI gives the conclusion.

2 HADOOP DISTRIBUTED FILE SYSTEM

HDFS is an open source implementation of Google File System (GFS) (Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, 2003). It is one of the core components of Hadoop and provides the basic function of file storage to the upper components such as MapReduce and Hbase. HDFS is a file system designed for storing very large files (typically 64M each block) with streaming data access patterns, running on clusters on commodity hardware (Tom White, 2009).

Li B. and Xu K..
A METHOD OF ADJUSTING THE NUMBER OF REPLICA DYNAMICALLY IN HDFS.
DOI: 10.5220/0003587005290533
In *Proceedings of the 13th International Conference on Enterprise Information Systems* (SSE-2011), pages 529-533
ISBN: 978-989-8425-53-9

The HDFS cluster has a master-slave architecture and primarily consists of one Namenode and several Datanodes. This architecture ensures all the data will not flow through the Namenode and reduce the workload of Namenode. This is a single point of failure for an HDFS installation. If the Namenode goes down, the file system is offline. Normally there is a secondary node as the backup of Namenode.

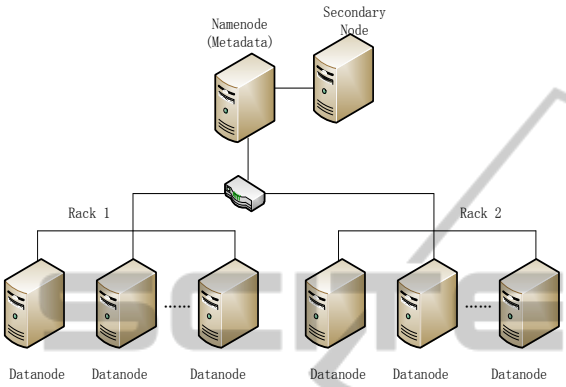

Figure 1: HDFS Architecture.

- Namenode

Namenode is the primary node in HDFS. It manages the file system metadata instead of data itself. Namenode has to maintain the directory tree, a mapping between HDFS files, a list of blocks and the location of those blocks. And more, Namenode provides management and control service.

Namenode will store the Metadata of directory and file in a binary file called fsimage which is the most important part related to the procedure of read and write. Every operation before the fsimage saving will be recorded in editlog. When the editlog reach certain size or a certain time passed, Namenode will refresh the Metadata to fsimage. That's how Namenode ensure the security of Metadata information of HDFS.

The form of fsimage shows in figure 2:

imgVersion (int)		namespaceID (int)		numFiles (long)		genStamp (long)						
path (String)	replica (Short)	atime (long)	atime (long)	blocksiz (long)	numBlock (int)	nsQuota (long)	dsQuota (long)	username (String)	group (String)	perm (Short)		

Figure 2: Fsimage information.

When a Datanode start and join in the HDFS, it will scan its disk and report the block information to Namenode. Namenode keeps this information in its memory with the datanode information. And then, the chart of block to Datanode list is constructed. This chart information is saved in a data organization called BlockMap. The BlockMap has the information of Datanode, block and replica.

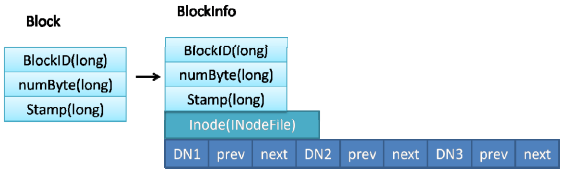

Figure 3: BlockMap.

The BlockMap is showd in figure 3. There are 3 replica of a block. Each replica information is a tree-element group. DN means which Datanode has the replica; prev means the previous BlockInfo quota of this block in the Datanode; next means the next BlockInfo quota of this block in the Datanode. The number of this three-element group is as same as the number of replica.

- Datanode

DataNode stores and manages the actual data. Each file was split into one or more blocks and these blocks are stored in a set of Datanode. We can simply think Datanode stores the block ID, the content of block and the mapping relationship.

A HDFS cluster normally has thousands of Datanode. These Datanode communicate to Namenode periodically.

- Replica

HDFS achieves reliability by replicating the data across multiple hosts. Each block has one or more replica in other Datanodes and each Datanode has one replica of a file at most. With the default replication value 3, data is stored on three nodes: two on the same rack, and one on a different rack. Generally, the node on the different rack has lower chance to fail.

Developers can set the parameter to determine the number of replica they want. More replicas mean the better reliability and more disk space wasted. Each replica of block is recorded in Namenode. Client could access the nearest Datanode to read/write the block. And if one replica was changed, HDFS will automatically change all these replicas in other Datanode through pipe mechanism.

- Block Read Procedure

At first, client sent a message to Namenode through RPC to get a block list of specific file and the address of Datanode where all the replicas located. And then, the client connected those Datanode and sent a requests of those blocks to build the link. After the link was built, client read those Blocks one after one.

- Flow
  - HDFS has a single point failure problem: if

the Namenode dead, all the system would go off-line;

- HDFS is designed for big files (typically the size is GB or TB) and not work well for small files: the I/O mechanism is not fit for small files, and Namenode keep all the metadata in its memory, the size of memory determines the number of files HDFS can keep.
- Balancing portability and Performance: HDFS is written in Java and designed for portability across heterogeneous hardware and software platforms, there are architectural bottlenecks that result in inefficient HDFS usage;
- HDFS has a hot spot problem: HDFS has the same replica number for all the files, if files access frequency was different, some were welcomed and some cared by nobody, there will be hot spot. If an access burst happened, the existing replica may not satisfy the users. The hot spot problem will cause the local network congestion and reduce the throughout rate of the whole system.

- Solutions
  - Single point failure problem: we can set a secondary not to be the backup of Namenode. And Feng Wang (Feng Wang, Jie Qiu, Jie Yang, 2009) etc.. have done a great job to solve this problem through metadata replication;
  - Small files problem: Grant Mackey and Saba Sehrish (Grant Mackey, Saba Sehrish, Jun Wang, 2009) provide a solution to improving metadata management for small files in HDFS; Xuhui Liu and Jizhong Han etc (Xuhui Liu, Jizhong Han, Yunqin Zhong, Chengde Han and Xubin He, 2009). introduce a solution to combine small files into large ones to reduce the file number and build index for each file;
  - Balancing portability and Performance: Jeffrey Shafer and Scott Rixner (Jeffrey Shafer, Scott Rixner, and Alan L. Cox, 2010) have done a great job to investigate the root causes of these performance bottlenecks in order to evaluate tradeoffs between portability and performance in HDFS;
  - Hot spot problem: this paper introduces a system-level strategy by given each Block a independent replica configuration; when the reading requests became higher than system capacity, the Namenode will increase the replica number of specific Block.
- Hot spot Bottleneck analysis

The user's access demand to each file is different. Some files have been visited a lot and some others are silence. Moreover, those hot files are not specified all the time. Sometime, natural accidents and social incidents such as earthquake and Olympics will create new hot files. And sometimes social trend make these old silence files become welcomed. If huge number of clients visit the same block at the same Datanode, there will be hot spot. Because of the limited of host hardware capability and network throughput, the access performance will decrease sharply and the user experience will become unacceptable. Unfortunately, the Namenode and system operator is hard to know which file or block will get visited frequently. There is a need of a mechanism to tell if a block has become a hot spot and its location.

Hadoop takes the multi-replica way to deal with parallel reading of the same block. The number of replica was set in the fsimage we have already introduced before the whole cluster began to work and the configuration is valid in the whole system. All the Datanodes would take the same configuration, and all Blocks has the same number of replica.

So, there will be two problems: the first one, when the reading requests exceed more than the expectation, the number of replica configuration is hard to meet the demand; and the second, if we set the number of replica much bigger, there will be huge waste of hard-drive space of the system obviously. If the number of replica plus from 3 to 6, there is half space left.

Because the Hadoop takes multi-thread and NIO to deal with parallel reading problem, the bottleneck of the distributed system is hard-drive IO performance. We use a 7200rpm and 133M/S of external transfer rate in order to void the influence of network throughout performance to the experiment. When the number of reading requests was lower or equal to the number of replica, the average reading speed is 113.7M/S. Consider of the difference of theory data and real experimental environment, the result is acceptable. And when we set the number of reading requests as much as twice of the number of replica, the experiment data shows the average read speed S is 52.8M/S.

$$S_{average} = (F \times n) \div T$$

The $S_{average}$ represents the average reading speed, the F represents the file size, the n represents the number of requests and the T means the total time. We can see that the average reading speed was roughly liner downward.

According to the analysis, we can assume that the reading requests to the specific file increase sharply lead by some social incidents, the system cannot support the huge traffic and the user experience will become unacceptable. That's why we should adjust the number of replica of each file. And at the same time we also need to monitor the number of requests. If the number of request exceeds the threshold, we should increase the number of replica. And if the number of request is less than the threshold, the system will automatically reduce the number of replica.

In section III, we introduce a method to solve those problems.

3 DESIGN DETAIL

Normally, if a client wants to read a file, it will connect the NameNode first and find out which DataNode has the file. So we can simply record the number of connection to tell which DataNode has become hot spot.

3.1 New Variable in Namenode

At first we change the replications in fsimage to a minimal number of replica. It means the system will read this value when it checks the replica number of Block.

Second, we set two new variables named numReplica and connectCounter after the BlockID in BlockMap which shows in Figure 4.

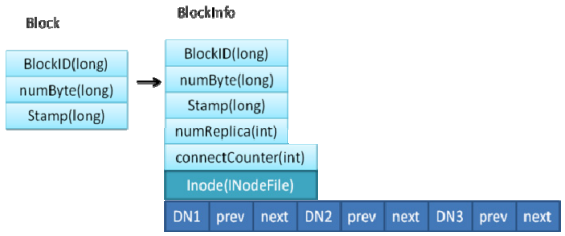

Figure 4: BlockMap with new variables.

At the very beginning, the numReplica equals to the minal number of replications in fsimage. And the system could set the number of replication for each Block according to the numReplica. The connectCounter is a parameter which shows how many clients are read the Block at the same time. When there is a new request to read a file from a client, the connectCounter of those Blocks which belong to this file will plus one. Blocks from same file have the same numReplica and connectCounter.

3.2 DataNode Response

In order to prevent the connectCounter keep increasing, we need to make client send a message to Namenode when reading is finished. And after Namenode received this message, the connectCounter should reduce by one.

3.3 Minimal Value

We set the minimal number of replica in the fsimage, which means each Block's number of replica cannot be less than this value.

3.4 Add and Reduce Replica

When the connectCounter is bigger than the threshold, Namenode will trigger the procedure of increasing replica. This procedure is the original one in Hadoop. Namenode will instruct Datanode get a copy of the Block from other Datanode, and it will update the BlockMap.

When the connectCounter is less than the threshold, Namenode will start the procedure of reducing replica. Namenode will instruct the Datanode move the Block into /trash directory, and update the BlockMap also.

4 EXPERIMENT

We compare the performance between the original system and the new design one. Each of them run on the same cluster with one Namenode and twelve Datanodes, all the machines are configured with dual 2.0 GHZ processor, 1GB memory, two 80GB disks(7200rpm with 133M/S) and a 1000 Mbps Ethernet connection switch. The operating system is ubuntu10.04. The version of Hadoop is 0.20.2 and java version is 1.6.0.

There are three kinds of file which size are 64Mb, 128Mb and 256Mb in the system. It has been divided into 1, 2 and 4 Blocks. The size of Block is typically 64Mb. In order to ensure the static precisely, those Blocks cannot be stored in a same Datanode. We try to make each Datanode have the same number of Block. We have ten clients reading those Blocks at the same time. The default number of replica is 3.

From the experiment, we got two groups of results. One is from the control group which takes the original system and the other result is from an optimization system. The actual experiment result shows in Table I, Table II and figure 5.

Table 1: Experiment result from the original system.

	128Mb	256Mb	512Mb
Total Time Cost(s)	2.61	3.37	8.45

Table 2: Experiment result from the optimization system.

	128Mb	256Mb	512Mb
Total Time Cost(s)	1.97	3.39	8.50

There are 2 blocks in one Datanode at most in the 512Mb condition. Theoretically, the total time cost should approximately be as much as twice of the time cost in the condition of file with 256Mb. However, we can see from the experiment result it's much higher than that. The reason is HDFS reads Blocks of the same file in order. If the client hasn't finish Block 1, it cannot read Block 2 first.

Moreover, in the condition of file with the size of 256Mb, the total time cost for the optimization system is roughly equal to the time cost for the original system. That's because there is no more room for another replica. As a result, the replica adjusting procedure cannot be activated.

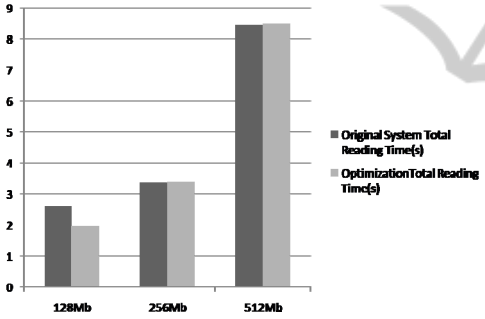

Figure 5: Total Time Cost Contrast.

And last, when there is enough space for new replica, the improvement of system performance is obviously. But the total time cost from the optimization system in the condition of 128Mb is not equal to 0.96s theoretically. Real number is higher than the ideal one. That's because the system detect there is less replica than the access requests and activated replica adjusting procedure. The time cost consisted of three parts. One is the normal reading time cost, and the second is the cost of communication between Namenode and Datanode, the third part is the cost of transfer Block between Datanodes.

5 CONCLUSIONS

HDFS takes replica of block to store large files and suffers performance penalty while some file became hot spot and a huge number of client send a request to Namenode in order to read this file. In this paper, we optimize the HDFS replica strategy and improve the system performance by adjusting replica number dynamically. So the file in hot spot could get more replicas to deal with file access. We also compare the optimization system to the original system in 3 ways: full of space for more replicas, the number of Datanode is equal to the number of block and the number of block is more than Datanode. We can see, in the condition 1, the optimization could improve the system performance by 25 percent.

ACKNOWLEDGEMENTS

This work is supported by the National Key project of Scientific and Technical Supporting Programs of China (Grant Nos.2008BAH24B04, 2008BAH21B03, 2009BAH39B03); the National Natural Science Foundation of China(Grant No.61072060); the Program for New Century Excellent Talents in University (No.NECET-08-0738); Engineering Research Center of Information Networks. Ministry of Education.

REFERENCES

Apache Hadoop. http://hadoop.apache.org/

J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters", In *OSDI'04: Proceedings of the 6th Symposium on Operating Systems Design & Implementation*, pages 10–10, 2004.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System", In *SOSP'03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, page 29–43, New York, NY, YSA, 2003. ACM Press.

Tom White, "*Hadoop: The Definitive Guide*", O'Reilly Press, 2009

Feng Wang, Jie Qiu, Jie Yang. "Hadoop High Availability through metadata Replication", *CloudDB'09*, November 2, 2009, Hong Kong, China.

Grant Mackey, Saba Sehrish, Jun Wang, "Improving Metadata Management for Small Files in *HDFS", IEEE International Conference on Cluster Computing and Workshop*, 2009.

Xuhui Liu, Jizhong Han, Yunqin Zhong, Chengde Han and Xubin He, "Implementing WebGIS on Hadoop: A Case Study of Improving Small File I/O Performance on HDFS", *IEEE, International Conference on Cluster Computing and Workshops*, 2009

Jeffrey Shafer, Scott Rixner, and Alan L. Cox, "The Hadoop Distributed Filesystem: Balancing Portability and Performance", *IEEE International Sympossium on Performane Analysis of Systems & Software (ISPASS)*, 2010.