LETTER

An Efficient Block Assignment Policy in Hadoop Distributed File System for Multimedia Data Processing

Cheolgi KIM[†a)], **Daechul LEE**[†], **Jaehyun LEE**[†], ***Nonmembers***, ***and*** **Jaehwan LEE**[†b)], ***Member***

SUMMARY Hadoop, a distributed processing framework for big-data, is now widely used for multimedia processing. However, when processing video data from a Hadoop distributed file system (HDFS), unnecessary network traffic is generated due to an inefficient HDFS block slice policy for picture frames in video files. We propose a new block replication policy to solve this problem and compare the newly proposed HDFS with the original HDFS via extensive experiments. The proposed HDFS reduces network traffic, and increases locality between processing cores and file locations.

key words: *Hadoop, Hadoop distributed file system, video processing, group of pictures*

1. Introduction

In recent years, distributed processing systems for large data processing have evolved rapidly. MapReduce frameworks [1] in Hadoop Distributed File Systems (HDFS) [2] and Spark [3] are examples. These frameworks are used not only for processing text data, but also for processing unstructured data, such as images and videos. HDFS divides a file into blocks of predetermined size (the default value is 64MB), and distributes them to the data nodes of the cluster. When processing these blocks in the cluster, each node processes the data in a way that leverages the locality using the MapReduce framework.

Meanwhile, codecs, tools for encoding/decoding video, are further improving processing by increasing the compression rate. A codec with a high video compression rate is capable of minimizing a file size without losing image quality. The MPEG-4 AVC codec is one of the most advanced codecs to date. MPEG4 video consists of the I-frame, which can be decoded by itself, and the B-frame and P-frame, which depend on neighboring frames, like the I-frame, for decoding. The Group of Pictures (GOP) in MPEG-4 is a group of I-frames, B-frames, and P-frames that can be fully decoded by themselves.

To process large amounts of video data, there have been many attempts [4], [5] to use clustered computers for distributed processing. However, when processing video using HDFS and MapReduce, there is a problem if the GOP is at the boundary of two HDFS blocks that are stored on different nodes. Due to this discrepancy between semantic video data and HDFS block slicing policy, a network transfer is needed to get information from the I-frame in remote nodes, which negates the design advantage of HDFS locality policy. In this paper, we propose a new block replication method to solve this problem.

In Sect. 2, we discuss video transcoding and HDFS, forming the background for this study. In Sect. 3, we describe the problem that occurs in HDFS video processing, and suggest a new solution. For evaluation, we have modified the original HDFS codes, and compare it with the original HDFS when processing videos in Sect. 4. Finally, we offer conclusions in Sect. 5.

2. Background

2.1 Video Transcoding

As stated, the GOP consists of multiple picture frames interdependent upon each other, as shown in Fig. 1. To decode a P- or B-frame in a GOP, other frames in the same group are required. If a system has only a part of the GOP, it cannot have the decoded pictures until the others are obtained. P-frames are encoded by referring to pre-existing I- or P-frames in the GOP, while B-frames depend on neighboring frames in the front and back of the GOP.

2.2 Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) supports scalability and availability to storage in the commodity servers. In HDFS, a file is divided into blocks, and each block is replicated and stored across multiple machines. HDFS has a master/slave architecture; with the NameNode performing the role of the master, and DataNodes acting as slaves. A NameNode manages the filesystem namespace and the metadata for files and directories. A DataNode stores real data blocks. Client applications that need a file in the

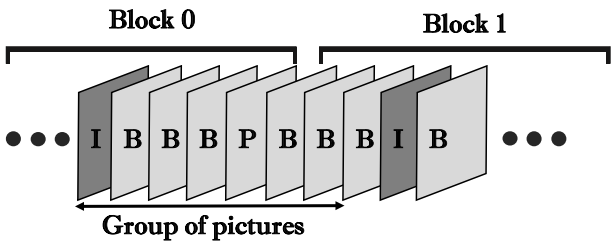

Fig. 1 The structure of the group of pictures.

Manuscript received January 22, 2019.
Manuscript publicized May 21, 2019.
[†]The authors are with Korea Aerospace University, Goyang-city, Korea.
a) E-mail: cheolgi@kau.ac.kr
b) E-mail: jlee@kau.ac.kr (Corresponding author)
DOI: 10.1587/transinf.2019EDL8016

HDFS can directly communicate with the DataNode after the NameNode transfers metadata containing the location of other corresponding data blocks.

3. Motivation and Methodology

3.1 Block Replication Policy and the Problem

As mentioned above, the HDFS replicates and stores blocks on multiple machines to provide fault tolerances. The default replication factor is set to 3. The NameNode determines all procedures for the replication of the blocks. If the replication factor is set to 3, the default value, the first replica is stored on the DataNode in the local rack, and the two others are stored on the remote rack. This policy prevents losing data when the entire rack fails, and allows the use of the multiple racks' bandwidth.

But there is a problem with this policy when processing continuous data like compressed video files. Figure 1 shows a GOP lying at the boundary of two blocks, with the boundary between the logical unit GOP and the physical unit block out of alignment. Such a mismatch may cause unnecessary network overhead. Suppose that the client processes the frames in the beginning of Block1. In order to process the B-frames at the beginning of Block1, the metadata from the I-frame in Block0 is needed. If Block0 and Block1 are stored in the same DataNode, the process running on that DataNode simply reads the necessary information in Block0 from local storage. However, since the policy of block replication in HDFS is arbitrary, the probability that successive blocks are stored in the same DataNode becomes lower as the cluster size increases. In most cases, such continuity is not guaranteed. In this case, the DataNode for Block0 requests information about the block location from the NameNode. After receiving the block location information from the NameNode, the DataNode for Block0 sends the necessary data to the DataNode that is requesting information in Block0. This process is what increases network overhead. This may not be a problem if there is a small number of users processing compressed videos, but this network overhead may make the network a bottleneck point, and increase the processing latency. This issue becomes more problematic as the unit of GOP becomes larger, which is a recent issue.

3.2 Our Methodology

As described in Sect. 3.1, the block replication policy in HDFS does not utilize locality when dealing with data in which continuity is important, thereby causing network traffic. In this paper, we propose a method to reduce network traffic and improve job performance by maintaining data continuity when processing compressed moving pictures.

The new block replication policy we propose stores consecutive blocks in each DataNode in the same quantity as the replication factor. Figure 2 demonstrates this policy at a replication factor set to 3; allowing the blocks stored in the

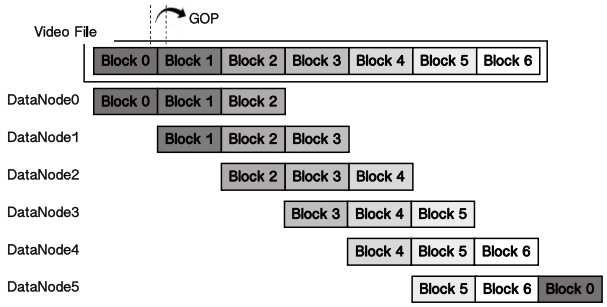

Fig. 2 Block location illustration for the proposed replication policy.

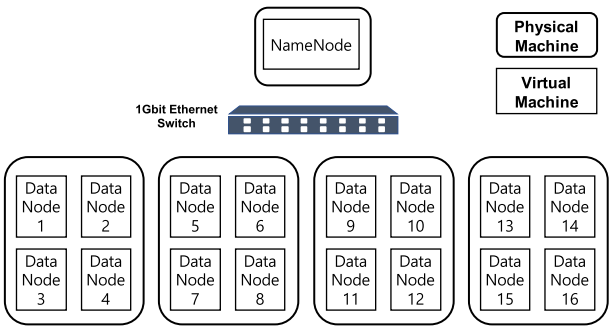

Fig. 3 Testbed cluster for the experiment.

Table 1 Hadoop cluster configuration parameters on virtualbox.

Properties	Values
No. of Nodes	17 (1 NameNode and 16 DataNodes)
Virtual Box Host Machines	Intel Xeon E3-1240V3, 8 cores, 4GB RAM CentOS 6.7, 64bit, 256GB SSD
Virtual Box Guest Machines	1Gbit Ethernet, 1 core, 768MB RAM CentOS 6.9, 30GB Disk
Softwares	Apache Hadoop 2.4.1, JVM - OpenJDK 1.7.0
HDFS parameters	Replication factor = 3 (default) Block size = 64MB (default)

DataNodes to form a ring structure. For each DataNode, no network traffic is required except when processing the first frame of the block at the beginning of the node.

We modified the HDFS source code to replicate and store the blocks as described above. In the next section, we will compare and analyze the original HDFS and our proposed replication policy through experimentation.

4. Experiments

Figure 3 and Table 1 shows the testbed for the experiments. We set up the 17 nodes in our cluster running on virtual machines using Virtualbox.

The goal of the experiments is to compare the resources used to process the compressed video, focusing on network traffic. The data to be processed is a 1024MB MPEG-4 video file. It is divided into 16 blocks, and each DataNode contains three blocks. The default block size is 64MB and replication factor is set to 3.

In each of the 16 virtual machines, the Java process reads the local block and the previous block. If the previ-

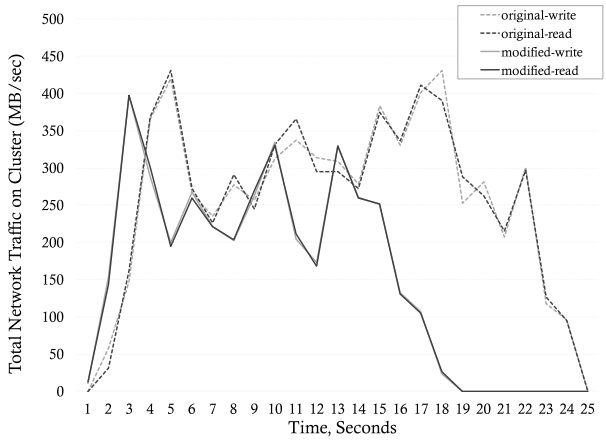

Fig. 4 The I/O test results for 8GB data.

ous block is stored locally, only a small amount of network traffic occurs, but in other cases, the NameNode and the DataNode for that block generate additional network overhead. Note that the amount of the data that each process reads is 128MB.

Figure 4 shows the experiment results. In the case of the original HDFS, all consecutive blocks are not adjacent. It took 25 seconds on average to complete the entire job, all 16 DataNodes generated network traffic, and the network became a bottleneck. At 2 seconds, the network transfer rate reached 450MB/sec, including both read and write. Because the four physical machines were connected to a 1 Gbit Ethernet switch, the total aggregated network bandwidth was 512MB/sec. Since network I/O occurs at the same time, we can see the bottleneck in the network bandwidth from 4 seconds to 5 seconds and 17 seconds to 19 seconds.

In the case of modified HDFS, for each DataNode, the blocks are adjacent to each other in the same quantity as the block replication factor. In this configuration, the network did not become a bottleneck, and took 19 seconds on average to complete the entire job.

We also experimented with other conditions. We resized the file size and block size. We measured the total network traffic used by the job and divided the value of the modified Hadoop by the value of the original Hadoop to obtain a normalized value for each case. Figure 5 shows the experiment results with various conditions. As you can see in the graph, the values are about 0.5 to 0.6, which means that when using the modified HDFS, it occurs 50% to 60% of network traffic than the original one. This graph also shows that our system works well in various conditions.

As we have seen above, when we process interdependent data such as compressed video in the manner presented in this paper, network usage is lower than processing according to the original HDFS policy. As a result, when network bandwidth is not sufficient, job execution time can be shortened by avoiding a network traffic bottleneck.

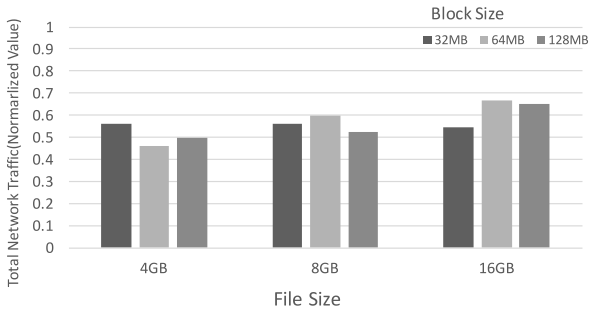

Fig. 5 The experiment result with resized block size and file size.

5. Conclusion

In this paper, we have seen that processing video data on HDFS creates network overhead due to the metadata dependency of the video file. To reduce this network overhead, we proposed a new block replication policy. Our experiments showed that storing consecutive blocks on the same node, in the same quantity as the block replication factor, significantly reduced network traffic when processing video data in the HDFS. Considering that network bandwidth is a limited resource in the distributed environment, if there were an alternative HDFS that supported this replication policy, job performance might improve in situations where data processing continuity is important.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2014R1A1A1008439, NRF-2019R1H1A2039658, NRF-2015M3C4A7065646) and partly supported by the GRRC program of Gyeonggi province (No. GRRC-KAU-2018-B01, "Study on the Video and Space Convergence Platform for 360VR Services"), ITRC (Information Technology Research Center) support program (IITP-2019-2018-0-01423) and ICT R&D program of MSIP/IITP (B0101-16-0663, Safety-critical Distributed Modular SW Platform).

References

[1] J. Deen and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol.51, no.1, pp.107–113, 2008.
[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.
[3] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," HotCloud 10, 2010.
[4] R. Pereira, M. Azambuja, K. Breitman, and M. Endler, "An architecture for distributed high performance video processing in the cloud," 2010 IEEE 3rd International Conference on Cloud Computing, pp.482–489, 2010.
[5] F. Lao, X. Zhang, and Z. Guo, "Parallelizing video transcoding using Map-Reduce-based cloud computing," 2012 IEEE International Symposium on Circuits and Systems, pp.2905–2908, 2012.