2013 IEEE International Conference on Cloud Computing Technology and Science

Extensible Video Processing Framework in Apache Hadoop

Chungmo Ryu, Daecheol Lee, Minwook Jang, Cheolgi Kim
Dept. of Computer and Information Engineering
Korea Aerospace University
Gyeonggi, Korea
Email: {fbcndah, rod8902, mwjang86, cheolgi}@gmail.com

Euiseong Seo
Dept of Software
Sungkyunkwan University
Suwon, Korea
Email: euiseong@gmail.com

***Abstract*—Digital video is prominent big data spread all over the Internet. It is large not only in size but also in required processing power to extract useful information. Fast processing of excessive video reels is essential on criminal investigations, such as terrorism. This demo presents an extensible video processing framework in Apache Hadoop to parallelize video processing tasks in a cloud environment. Except for video transcoding systems, there have been few systems that can perform various video processing in cloud computing environments. The framework employs FFmpeg for a video coder, and OpenCV for a image processing engine. To optimize the performance, it exploits MapReduce implementation details to minimize video image copy. Moreover, FFmpeg source code was modified and extended, to access and exchange essential data and information with Hadoop, effectively. A face tracking system was implemented on top of the framework for the demo, which traces the continuous face movements in a sequence of video frames. Since the system provides a web-based interface, people can try the system on site. In an 8-core environment with two quad-core systems, the system shows 75% of scalability.**

I. Introduction

As smart phones and digital surveillance video systems flourish, video becomes one of the dominant big-data applications. From surveillance video, we may want to identify and track the suspects. In a user-video database, automatic face recognition can be an attractive application. Video data are not just large in size, but also in required processing power. To extract meaningful information from a video, such as facial signatures, massive process per image is needed, while the entire video reel may have an excessively long sequence of images. However, such processing must be timely performed in mission-critical applications. For instance of 2011 Vancouver ice hockey final riot, the police inspected 1600 hour video, and a million images to identify the prime suspects. In such cases, investigation must be completed in a couple of days in general. Consequently, Video data processing is highly suitable to adopt a cloud computing environment, such as Apache Hadoop [1], which conveniently parallelize the processing. In this demo, we present a parallel video processing framework in the Apache Hadoop cloud-computing environment.

This work is supported by the the Gyeonggi Regional Collaboration Research Center [GRRC-KAU-2013-B02]

Apache Hadoop is the most popular big-data cloud-computing environment, whose source code is freely accessible in public. It implements a distributed fault-tolerant file system, called HDFS (Hadoop Distributed File System) and the MapReduce distributed processing pattern. Our video processing framework is also based on both HDFS and MapReduce pattern. It reads a video reel from HDFS, and distributes the reel in the cloud for a process using MapReduce pattern. To process the video, it first decompresses the stream using modified FFmpeg library, and processes the uncompressed image sequences with the OpenCV library to extract essential information. Since FFmpeg and OpenCV are in C language without a concern of Hadoop environment in Java, extensions of the libraries were required, which are found to be nontrivial. In our best knowledge, it is the first extensible video-processing framework that modifies FFmpeg library to optimize for the Hadoop environment, even though there are a couple of video transcoding systems in Hadoop without off-the-shelf use of FFmpeg executable [2]–[4].

To demonstrate our framework, we implemented a face tracking system, in which each sequence of the same person's faces in a continuous motion is collected and grouped automatically. It can be one of basic features in facial information extraction from surveillance video data.

The paper is organized as follows. We first introduce the basic concepts and related work in video-related cloud systems in Section 2. In the following section, we explain our framework and technical challenges we met. Section 4 describes our demo system and evaluates experimental results. We conclude our work in Section 5.

II. Related Work

As mentioned earlier, there have been several video transcoders implemented in cloud environments. X264farm is one of the early systems, parallelizing a video encoding job in a distributed system [5]. It divides a video into group of pictures (GOPs) and distributes the work with load balancing. Mohohan is the first work that employs Hadoop and FFmpeg for a cloud video transcoder [2]. It connects multiple off-the-shelf applications to divide a video file into GOP units, transcode the pieces and combine

978-0-7695-5095-4/13 $31.00 © 2013 IEEE
DOI 10.1109/CloudCom.2013.153

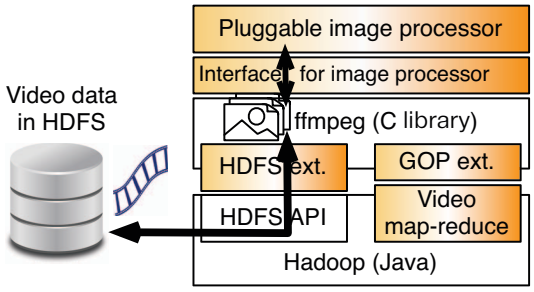

Figure 1. Architecture of our framework; Boxes with orange (or dark) shade is the ones that we implemented for the framework

them. Although it does not address optimizations, it was a nice collection of applications to rapidly construct a distributed transcoding system. Kim et al. also proposed a cloud transcoding system based on Hadoop and FFmpeg [4]. To use FFmpeg library in Hadoop, they use Xuggler, Java interfaces of FFmpeg library. It provides a cloud computing architecture that transcodes video data in a single map-reduce application. Yang and Shen also implemented a similar transcoding system with Hadoop and FFmpeg [3]. Garcia et al. implemented a prototype to measure the performance of such transcoding systems [6].

On the other hand, there are few video-processing systems implemented. One of such efforts is HP Labs' VideoToon system, which cartoonizes videos in a Hadoop environment. It uses HP's own cartoonizing engine, and is not intended for other video-processing extensions [7].

III. Our Framework

The purpose of our work is to provide a framework to develop a video image processing cloud system. Since FFmpeg and OpenCV are already combined in the framework, developers can easily extend it with their own custom image processing engine. Once video data is decoded into a sequence of images by the modified FFmpeg library, each image is passed to the image processor, possibly custom built. Currently, image processor is supposed to use OpenCV, the most popular image processing library, but need not bounded to it. Since OpenCV is implemented in C and C++, and so is FFmpeg, we made video processing interfaces in C and C++ environment for optimizations. A lot of efforts on our framework have been devoted to the interface between the Java-based Hadoop subsystem and C-based FFmpeg library. In this section, we briefly introduce each component and such challenges.

A. Overall Architecture

In our framework, Video processing is performed at *map* stage of the MapReduce pattern. At the reduce stage, the proccessed data can be refined.

Fig. 1 illustrates the architecture of our demonstration framework. The system has three layers: Hadoop subsystem,

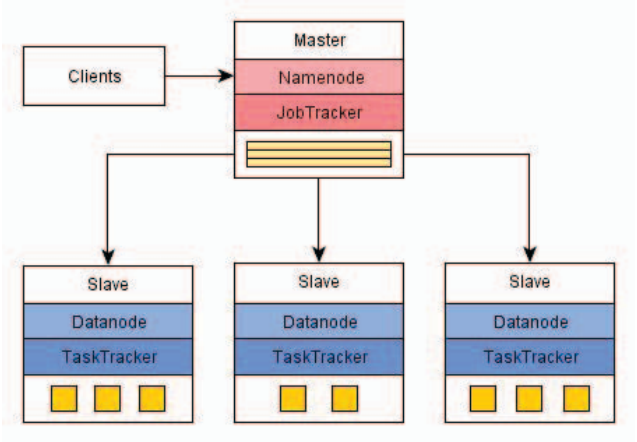

Figure 2. Namenode/Datanode and JobTracker/TaskTracker

FFmpeg subsystem and the image processor from the bottom to the top, between which the interfaces bridge. Hadoop manages the cloud system based on the MapReduce pattern. It decomposes the video processing job into GOP elements and distributes them to the cloud nodes. Each GOP element metadata is handed to the FFmpeg library. FFmpeg decodes a GOP into a sequence of images and passes them to the pluggable image processor one-by-one. The image processor processes the image sequence and manufactures processed images in structure.

B. How to access video files

To implement a video processing cloud system, the video decoder must be able to read a video file in HDFS. There can be a couple of ways to pass video data to the video decoder. First, Hadoop's MapReduce engine can read and pass the video data to FFmpeg. Second, Hadoop can just point out the section to decode to FFmpeg and FFmpeg directly reads the video data from HDFS. To optimize the performance, we chose the second option in spite that it needs an FFmpeg extension in video I/O subsystem to allow direct access to HDFS. The HDFS related portion of our framework is the left half of the framework in Fig. 1.

FFmpeg extension for HDFS was required to be developed because HDFS has only dedicated interfaces without a support of standard file APIs. Linux FUSE package was another option, which is a third party interface providing standard file APIs for HDFS. Since a Hadoop-based transcoder [2], uses Linux FUSE package, we measured the performance of the Linux FUSE package as an option for the framework. However, we decided to extend FFmpeg for a direct HDFS access, not using the FUSE package because it has significant performance losses. Our extension is not only faster than FUSE package, but also much stabler, we found.

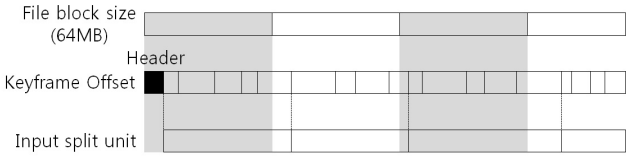

Figure 3. Splits divided by keyframe boundary near block

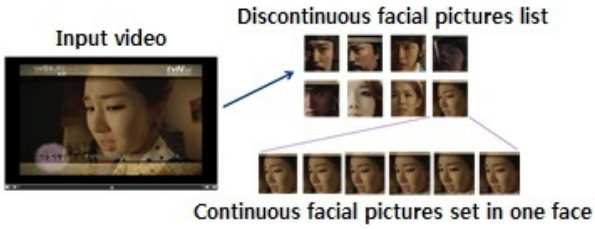

Figure 4. Output of our face tracing demo system

C. Distributed processing through MapReduce

Hadoop MapReduce, distributed processing engine in Hadoop, is a programming model to process big data in largely distributed computing nodes. Hadoop MapReduce has the Master node called JobTracker, and the slaves called TaskTracker. Their relationship is similar to the one between the NameNode and the DataNodes in HDFS. The relationship between NameNode, DataNodes, JobTracker, and TaskTrackers are depicted in Fig. 2

JobTracker gets a Job, usually a large problem from a client, and slices it into small sub-problems, each called `InputSplit`. Each `InputSplit` is allocated to a TaskTracker, which consecutively performs a map function for each entry in the split. Once a TaskTracker completes the allocated split, an additional split is assigned, if remained.

Performance of a MapReduce job considerably depends on how to divide a input file into `InputSplit`s. The size of a split is better to be smaller to maximize parallelism, but should be larger to minimize maintenance overheads. Consequently, it is important to find proper compromise for the size of a split.

Notice that an `InputSplit` cannot scale down to a picture frame because a frame in a video is dependent upon other nearby frames in decoding. Since the smallest independent decoding unit in video files is a GOP, the smallest split unit is also a GOP. Once a GOP is assigned to a TaskTracker as an `InputSplit`, it will decode the frames in the GOP and process each image sequentially.

When deciding the size of an `InputSplit`, it is better to be synchronized with a file block. Once it is synchronized, the JobTracker can distribute the split to the node that has the block as it can be read and processed within the node. Otherwise, the split must be requested remotely.

In this study, we adjust each `InputSplit` to be closely match with a file block. However, if each split is exactly the same as a file block, the split's starting point may not accord with a GOP starting point. Since a split cannot be finer than a GOP, a split boundary must be at a GOP boundary. Thereby, our framework divides an `InputSplit` at the GOP boundary just after the block boundary. With this strategy, each split will closely synchronize with a file block with minimal off-block data. As long as the JobTracker allocates a split to TaskTrackers that have the split locally, data-process locality can be established, and remote file access can be minimal only for the off-block data. How to divide each `InputSplit` is depicted in Fig. 3.

To perform such splits, the framework must be aware of the GOP boundary positions. However, FFmpeg does not provide APIs providing such information. Moreover, the framework must be able to process the video from the location in the byte precision. Hence, we extended FFmpeg to support GOP information retrievals and byte-precision video seek functionality to enable our design as shown in Fig. 1.

D. Image processing in OpenCV

OpenCV is the most popular image processing library to get numerical or meaningful information from high dimensional image data in the real world. OpenCV is composed of programing functions to process image data.

In our work, OpenCV is not a mendatory but recommended image processing environment. Once a video frame is decoded into a still image, the image is directly passed to the image processor in C or C++ native environment. If the Hadoop Java environment intervenes the image handover, there must be copies of images between the native and JVM memory spaces. Thereby, we put the image processing layer just on top of the FFmpeg video decoding layer to optimize performance. Besides our face-tracing demo system, other image processing applications can be applied because it has a pluggable architecture.

IV. Face tracing demo system in a video

We developed a demo application that extracts representative faces and traces continuous faces over the frames to show the performance and practicality of our framework.

The representative faces are listed for their own sets. The criteria of facial continuity are positions of faces and similarity of facial contrast distributions. As illustrated in Fig. 4, All the faces in the video are detected by the OpenCV face detector, and the continuous faces are traced and grouped. Because demonstration system for this demo program is produced with web-based interface, users can directly operate the system.

We measured the performance of the system by comparing the processing time of a multi-core cloud system with a single-core system.

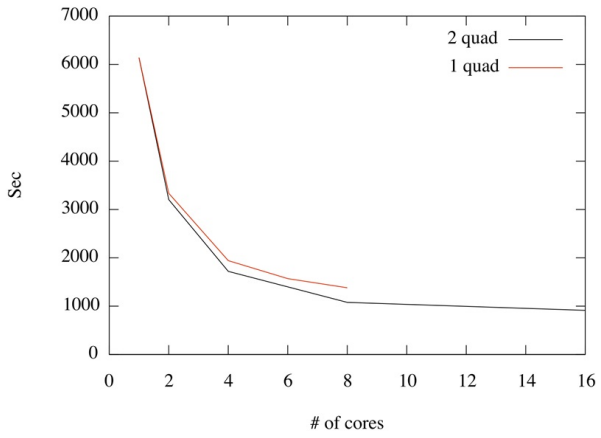

Figure 5. Graph of video-processing time

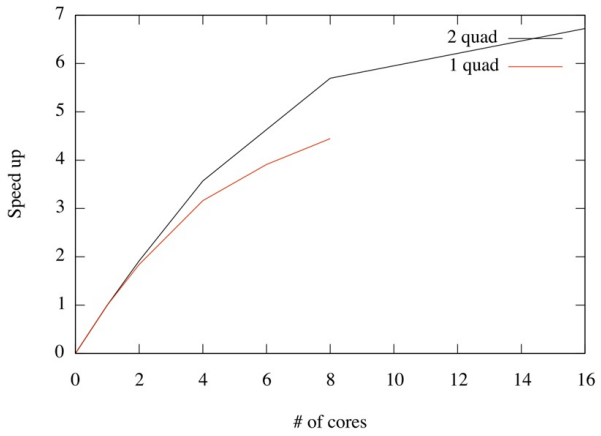

Figure 6. Graph of improving performance by increasing the number of cores

As a computing node, we employed two Mac Mini computers with a 2.3GHz i7 quad-core. Since the core supports hyper threading, we increased the number of cores to 16 logical cores, while there are only 8 physical cores. About 1GB Korean TV show was used for the measurement.

Fig. 5 and 6 show the performance results. Each graph shows the performance when one or two computers are used for processing. As long as the number of logical cores do not exceed physical cores, the system manages up to 75% of scalability. The performance losses seem to be from I/O overhead and coarse-grained `InputSplits`; at the end of process, many cores were found to be idle. Hyper threading was found to be not very effective in our system.

V. Conclusion

We implemented Hadoop-based video processig framework that is extensible for various video processing applications. To optimize the performance, we implemented the HDFS extension for FFmpeg to access video data in HDFS directly. Moreover, FFmpeg was also extended to provide GOP-related information to hadoop to divide a video file into appropriate splits, the unit of distributed processing.

We made a demo system that tracks faces with continuity through frames. The demo system was built on top of Web interface, for any user to test the system interactively. We found that our framework shows fairly good scalability in the performance evaluation. However, hyper threading does not seem to help the performance much.

References

[1] T. White, *Hadoop: the definitive guide*. O'Reilly, 2012.

[2] C.-H. Chen. Mohohan: An on-line video transcoding service via apache hadoop. [Online]. Available: http://www.gwms.com.tw/TREND_HadoopinTaiwan2012/1002download/C3.pdf

[3] F. Yang and Q.-W. Shen, "Distributed video transcoding on hadoop," *Computer Systems & Applications*, vol. 11, p. 020, 2011.

[4] M. Kim, Y. Cui, S. Han, and H. Lee, "Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 2, Mar. 2013.

[5] R. Wilson. x264farm. A distributed video encoder. [Online]. Available: http://omion.dyndns.org/x264farm/x264farm.html

[6] A. Garcia, H. Kalva, and B. Furht, "A study of transcoding on cloud environments for video content delivery," in *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*, ser. MCMC '10. New York, NY, USA: ACM, 2010, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/1877953.1877959

[7] H. Labs. Video Toon. [Online]. Available: http://www.hpl.hp.com/open_innovation/cloud_collaboration/cloud_demo.html

[8] [Online]. Available: http://fuse.sourceforge.net