INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 9(6):664-671, December, 2014.

Hadoop Optimization for Massive Image Processing: Case Study Face Detection

İ. Demir, A. Sayar

İlginç Demir
Advanced Technologies Research Institute
The Scientific and Technological Research Council of Turkey
ilginc.demir@tubitak.gov.tr

Ahmet Sayar*
Computer Engineerig Department
Kocaeli University, Turkey
*Corresponding author: ahmet.sayar@kocaeli.edu.tr

Abstract:
Face detection applications are widely used for searching, tagging and classifying people inside very large image databases. This type of applications requires processing of relatively small sized and large number of images. On the other hand, Hadoop Distributed File System (HDFS) is originally designed for storing and processing large-size files. Huge number of small-size images causes slowdown in HDFS by increasing total initialization time of jobs, scheduling overhead of tasks and memory usage of the file system manager (Namenode). The study in this paper presents two approaches to improve small image file processing performance of HDFS. These are (1) converting the images into single large-size file by merging and (2) combining many images for a single task without merging. We also introduce novel Hadoop file formats and record generation methods (for reading image content) in order to develop these techniques.
Keywords: Hadoop, MapReduce, Cloud Computing, Face Detection.

1 Introduction

In the last decade, multimedia usage has increased very quickly, especially as parallel to high usage rate of the Internet. Multimedia data, stored by Flicker, YouTube and social networking sites like Facebook, has reached enormous size. Today search engines facilitate searching of multimedia content on large data sets. So these servers has to manage storing and processing this much data.

Distributed systems are generally used to store and process large scale multimedia data in a parallel manner. Distributed systems have to be scalable for both adding new nodes and for running different jobs simultaneously. Images and videos are the largest set of these multimedia contents. So, image processing jobs are required to run in distributed systems to classify, search and tag the images. There are some distributed systems enabling large scale data storing and processing. Hadoop distributed file system (HDFS) [1] is developed as an open-source project to manage storage and parallel processing of large scale data.

HDFS parallel processing infrastructure is based on MapReduce [2], [3], [4] programming model that is introduced firstly by Google File System (GFS) [5] in 2004. MapReduce is a framework for processing highly distributable problems across huge data sets using a large number of computers (nodes), collectively referred to as a cluster.

The work presented in this paper proposes two optimization techniques to Hadoop framework for processing/handling massive number of small-sized images. This enables extreme parallel processing power of Hadoop to be applied to contents of massive number of image files. However, there are two major problems in doing so. First, image files need to be modeled in an appropriate format as a complete entity, and second, they need to be adapted to the mappers in order to utilize parallelization of Hadoop framework. As a solution to the first problem, an input file format called ImageFileInputFormat is developed. A new record generator class called ImageFileRecordReader is also developed in order to read/fetch the content of image and create whole image pixel data as an input record to MapTask [6].

Regarding the second problem, we develop two approaches. First approach is based on combining multiple small size files into a single Hadoop SequenceFile. SequenceFile is created by ImageFileRecordReader class, which is developed as an extension to Hadoop. Second approach proposes a technique to combine many images as a single input to MapTask without merging. This technique does not require special input file format as SequenceFile, so that images can be used as in their original format. To achieve this, we introduce novel image input format and an image record reader, which is MultiImageRecordReader class, to fetch the image content into image processing library. The architectural details are presented in section 3. These two approaches together with the naive approach are going to be applied on distributed face detection applications on images. The effectiveness of the proposed techniques is proven by the test cases and performance evaluations.

Remaining of this paper is organized as follows. Section 2 presents related works. Section 3 explains the proposed architecture. It covers extending Hadoops application programming interfaces to effectively manipulate images. Section 4 evaluates the performances of techniques on a sample scenario. Section 5 gives the summary and conclusion.

2 Related Work

HDFS is specialized in storing and processing large-size files. Small-size files storage and processing ends up with performance decrease in HDFS. NameNode is the file system manager in HDFS master node which registers file information as metadata. When using massive number of small-size files, the memory usage of Namenode increases so leading master node to be unresponsive for file operation requests from client nodes [7]. Moreover, number of tasks to process these files increases and Hadoop JobTracker and TaskTrackers, which initialize and execute tasks, have more tasks to schedule. In that way, total HDFS job execution performance decreases. For these reasons, storing and processing massive number of images require different techniques in Hadoop. Dong et al. propose two techniques for this problem given in [7] and [8]. In [7] they propose firstly, file merging and perfecting scheme for structurally related small files, and secondly, file grouping and perfecting for logically related small files. Their approach is based on categorization of files based on their logical or structural properties. In [8], they propose another similar approach on the same problem. They introduce a two-level perfecting mechanism to improve the efficiency of accessing small files, and use power point files as a use case scenario. On the other hand, we tackle this problem by introducing a new input file format and a new record generator class in order to read the content of images and create whole image data as an input record to MapTask. Hadoop [3] is based on a parallel programming paradigm MapReduce employing a distributed file system for implementation on very large clusters of low performance processors aimed at text based searching. Although it has been mainly utilized for textual data collections such as crawled web documents and web logs, later it has been adopted in various

types of applications. For example, it has been used for satellite data processing [4], bioinformatics applications [5], and machine learning applications [6].

There are a few example usages of Hadoop in image processing. These are mostly implementations of content-based image searching. Golpayegani and Halem [9] adopted Hadoop in satellite image processing. They propose a parallel text-based content searching. So, each image is annotated with some textual information after fetched by the satellites. Similarly Krishna et al. [10] proposes a Hadoop file system for storage and MapReduce paradigm for processing images crawled from the web. The input to the whole system is a list of image URLs and the contextual information aka the metadata of the image. Searching is done over the metadata of the images. The only challenge in such application is defining key and value pairs and as well as defining map and reduce functions. The other issues are handled by Hadoop core system. The architecture presented in [9] and [10] are called hybrid architectures, because they use text-based annotated data for searching and they access the result images by a URL defined in content data. Afterwards, they can process the image and redefine their metadata and save it with the current version. There is also another type of use cases of Hadoop as in [11]. Kocakulak and Temizel [11] propose a map reduce solution using Hadoop for ballistic image comparison. Firearms leave microscopic markings on cartridge cases which are characteristic to each firearm. By comparing these marks, it is possible to decide whether these two cartridge cases are fired from the same firearm or not. The similarity scores returned by the algorithm included similarity score for each part of the cartridge case and their weighted sum. All correlation results were passed by map tasks to the partitioner as key/value (k, v) pair. The key and the value constituted the id of the cartridge case and the similarity score object respectively. The work presented in this paper is different from [9] and [10] because they use hybrid model to search images, i.e., images are annotated with some textual information enabling content-based searching. In our model, we use images in their pure formats, and instead of text search we utilize advanced face detection algorithm by comparing pixel information in the images. In addition, since we are doing face recognition, we cannot cut the images (mapping) into small parts as in [11]. If we cut the images we can degrade the success of the system. In other words, we might possibly cut the image at a place where a face might be located. So, we keep the images as a whole and propose a different approach as given in Section 3.

3 Architecture: Hadoop Optimization for Massive Image Processing

3.1 Distributed Computing with Hadoop

HDFS is a scalable and reliable distributed file system consisting of many computer nodes. The node running NameNode is the master node and nodes running DataNode are worker nodes. DataNodes manage local data storage and report feedbacks about the state of the locally stored data. HDFS has only one NameNode but can have thousands of DataNodes.

Hadoop uses worker nodes as both local storage units of file system and parallel processing nodes. Hadoop runs jobs parallel by using MapReduce programming model. This model consists of two stages which are Map and Reduce whose input and outputs are records as <key, value> pairs. Users create jobs by implementing Map and Reduce functions and by defining the Hadoop job execution properties. After having defined, jobs are executed on worker nodes as MapTask or ReduceTask. JobTracker is the main process of Hadoop for controlling and scheduling tasks. JobTracker gives roles to the worker nodes as Mapper or Reducer task by initializing TaskTrack-

ers in worker nodes. TaskTracker runs the Mapper or Reducer task and reports the progress to JobTracker.

Hadoop converts the input files into InputSplits and each task processes one InputSplit. InputSplit size should be configured carefully, because InputSplits can be stored more than one block if InputSplit size is chosen to be larger than HDFS block size. In that way, distant data blocks need to be transferred over network to MapTask node to create InputSplit. Hadoop map function creates output that becomes the input of the reducer. So, the output format of the map function is same with the input format of the reduce function. All Hadoop related file input formats derive the FileInputFormat class of Hadoop. This class holds the data about InputSplit. InputSplit does not become input directly for the map function of the Mapper class. Initially, InputSplits are converted into input records consisting of <key, value> pairs. For example, in order to process text files as InputSplit, RecordReader class makes text lines of the file as an input record in <key, value> format where key is the line number and value is the textual data of each line. The content of the records can be changed by implementing another derived class from RecordReader class.

In distributed systems, the data to be processed is generally not located at the node that processes that data and this situation causes performance decrease in parallel processing. One of the ideas behind the development of HDFS is making the data processed in the same node where it is stored. This principle is called data locality which increases the parallel data processing speed in Hadoop [6].

Using massive number of small size files causes shortage of memory in master node due to increasing sizes of Namenode?s metadata file. Moreover, as the number of files increases, the number of tasks to process these files increases and the system ends up with workload increases in Hadoop?s JobTracker and TaskTrackers, which are responsible for initialization, execution and scheduling of the tasks. These might lead master node to be unresponsive for file operation requests from client nodes [7]. In brief, storing and processing massive number of images require different techniques in Hadoop. We propose a solution to this problem of Hadoop by an application of face detection.

3.2 Interface Design

In order to apply face detection algorithm to each image, map function has to get the whole image contents as a single input record. HDFS creates splits from input files according to the configured split-size parameter. These InputSplits become the input to the MapTasks. Creating splits from files causes some files to be divided into more than one split, if their file size is larger than the split-size. Moreover, a set of files can become one InputSplit if the total size of input files is smaller than the split size. In other words, some records may not be represented as the binary content of each file. This explains why new classes for input format and record reader have to be implemented to enable MapTask to process each binary file as a whole.

In this paper, ImageFileInputFormat class is developed by deriving the FileInputFormat class of Hadoop. ImageFileInputFormat creates FileSplit from each image file. Because, each image file is not splitted, binary image content is not corrupted. In addition, ImageFileRecordReader class is developed to create image records from FileSplits for map function by deriving Hadoop?s RecordReader class. In that way pixel data of images are easily fetched from Hadoop input splits into image processing tasks (map tasks). After that point, any image processing algorithm can

be applied to image content. In our case, map function of the Mapper class applies the face detection algorithm to image records. Haar Feature-based Cascade Classifier for Object Detection algorithm defined in OpenCV library is used for face detection [12]. Java Native Interface (JNI) is used to integrate OpenCV into interface. Implementation of map function is presented below. "FaceInfoString" is the variable that contains the information about detection properties such as image name and coordinates where faces are detected.

```
Class:          Mapper
Function:       Map
 Map(TEXT key(filename), BytesWritable value(imgdata), OutputCollector)
        getImgBinaryData_From_Value;
        convertBinaryData_To_JavaImage;
        InitializeOpenCV_Via_JNIInterface;
        runOpenCV_HaarLikeFaceDetector;
        foreach (DetectedFace)
                createFaceBuffer_FaceSize;
                copyFacePixels_To_Buffer;
                create_FaceInfoString;
                collectOutput:
                        set_key_FaceInfoString;
                        set_value_FaceImgBuffer;
        end_foreach
```

Hadoop generates names of output files as strings with job identification numbers (e.g.: part 0000). After face detection, our image processing interface creates output files as detected face images. In order to identify these images easily, the output file names should contain detected image name and detected coordinate information (eg: SourceImageName(100,150).jpg). ImageFileOutputFormat class is developed to store output files as images with desired naming. ReduceTask is not used for face extraction because each MapTask generates unique outputs to be stored in the HDFS. Each task processes only one image, creates output and exits. This approach degrades the system performance seriously. The overhead comes from initialization times of huge number of tasks.

In order to decrease the number of tasks, firstly, converting small-size files into single large-size file and process technique is implemented. SequenceFile is a Hadoop file type which is used for merging many small-size files [13]. SequenceFile is the most common solution for small file problem in HDFS. Many small files are packed as a single large-size file containing small-size files as indexed elements in <key, value> format. Key is file index information and value is the file data. This conversion is done by writing a conversion job that gets small-files as input and SequenceFile as output. Although general performance is increased with SequenceFile usage, input images do not preserve their image formats after merging. Preprocessing is also required for each addition of new input image set. Small files cannot be directly accessed in SequenceFile, whole SequenceFile has to be processed to obtain an image data as one element [14].

Secondly, combining set of images as one InputSplit technique is implemented to optimize small-size image processing in HDFS. Hadoop CombineFileInputFormat can combine multiple files and create InputSplits from this set of files. In addition to that, CombineFileInputFormat selects files which are in the same node to be combined as InputSplit. So, amount of data to be transferred from node to node decreases and general performance increases. CombineFileIn-

putFormat is an abstract class that does not work with image files directly. We developed CombineImageInputFormat derived from CombineFileInputFormat [15] to create CombineFileSplit as set of image. MultiImageRecordReader class is developed to create records from CombineFileSplit. This record reader uses ImageFileRecordReader class to make each image content as single record to map function (see technique in Fig.1). ImageFileOutputFormat is used to create output files from detected face images and stored into HDFS.

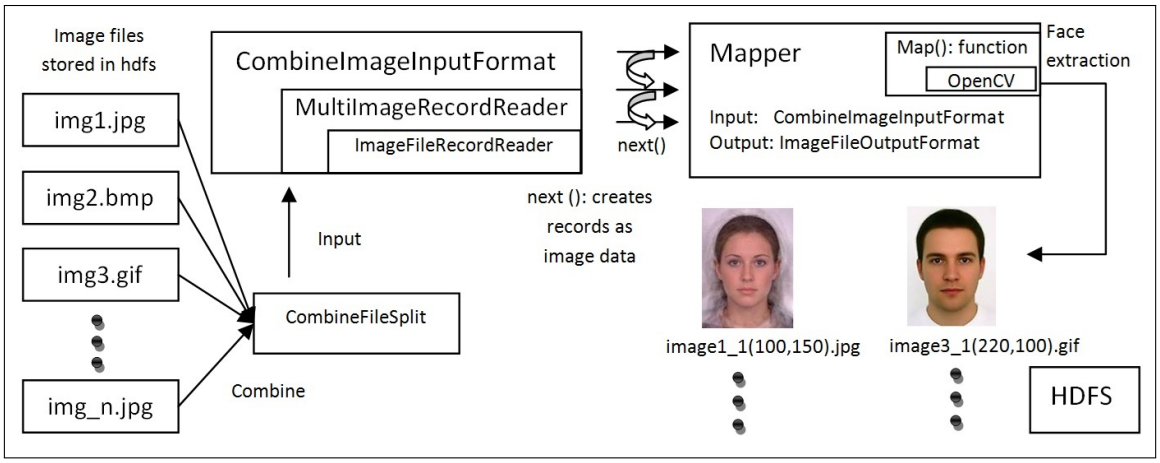

Figure 1: Combine and Process Images Technique

4 Performance Evaluations

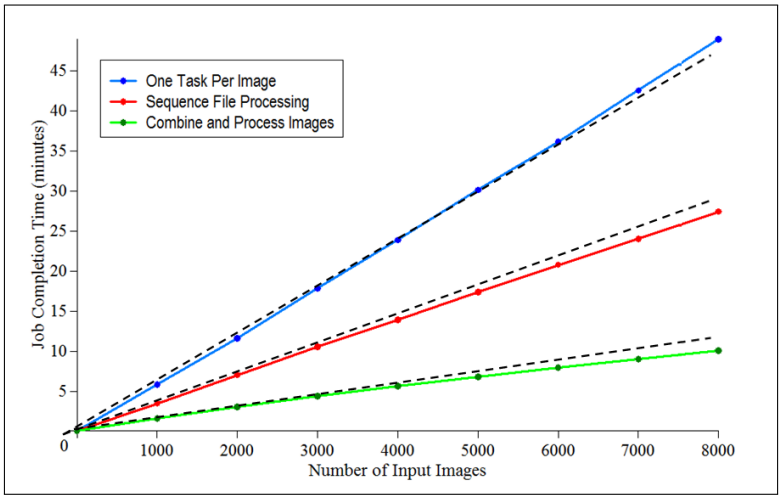

Figure 2: Performance Comparison

To test the system and evaluate the results, we have set up an HDFS cluster with 6 nodes. Face detection jobs are run on a given set of image files on the cluster. HDFS cluster is set up with 6 nodes to run face detection jobs on image sets. Each node has a Hadoop framework installed on a virtual machine. Although virtualization causes some performance loss in total execution efficiency, installation and management of Hadoop become easier by cloning virtual machines. MapTasks require large dynamic memory space when map-function for the image

processing executes. Default Java Virtual Machine (JVM) heap size is not enough for large size images. So, maximum JVM size for Hadoop processes is increased to 600 Mb.

Five different small size images are used as input files. Distribution of the images according to file sizes are preserved in input folders. The images in the input folders went through the face detection job with the three types of approaches in HDFS. These are (1) one task per image brute-force approach (for comparison only), (2) SequenceFile processing approach and (3) combine and process images approach (see performance results in Fig.2).

5 Conclusion

The effectiveness of the proposed technique has been proven by the test cases and performance evaluations. As Figure 2 shows, the proposed approach, combine images and then process, has become the most effective method in processing image files in HDFS. The SequenceFile processing is slower than the combining technique due to the fact that CombineImageInputFormat enforces creation of InputSplits by combining images in the same node. Additionally, in SequenceFile approach, InputSplits to be processed in MapTask does not always consist of datablocks in the same node. So some datablocks may be transferred from other storage node to MapTask node. Extra network transfer causes performance loss in total job execution. On the contrary, SequenceFile approach has better performance against the task per image approach, because small number of input files decreases number of created tasks. In that way, job initialization and bookkeeping overheads of tasks are decreased.

The slope of the job completion time curve for Task per image approach has increased as number of input images increases. But slopes of curves of the other two techniques have slightly decreased by increasing number of input images, because task per image approach causes heavy burden on initialization and bookkeeping by increasing number of tasks. On the contrary, number of tasks is not increased as proportional to the number of images in SequenceFile and combine images techniques. A small number of tasks has been able to process more images when the number of input images is increased.

Consequently, image processing like face detection on massive number of images can be achieved efficiently by using the proposed I/O formats and record generation techniques for reading image content into map tasks are discussed in this paper. We also explained the inner structure of map tasks to read image pixel data and process them. In the future, we plan to enhance and apply the proposed technique on face detection in video streaming data.

Bibliography

[1] http://hadoop.apache.org/.

[2] Berlinska, J.; M. Drozdowski. (2011); Scheduling Divisible MapReduce Computations, *Journal of Parallel and Distributed Computing*, 71(3): 450-459.

[3] Dean, J.; S. Ghemawat. (2010); MapReduce: A Flexible Data Processing Tool, *Communications of the ACM*, 53(1): 72-77.

[4] Dean, J.; S. Ghemawat. (2008); MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, 51(1): 1-13.

[5] Ghemawat, S.; H. Gobioff.; S. T. Leung.(2003); The Google File System, *Proceedings of the 19th ACM Symposium on Operating System Principles*, NY, USA: ACM, DOI:10.1145/945445.945450.

[6] White, T. (2009); The Definitive Guide. 2009: O'Reilly Media.

[7] Dong, B.; et al. (2012); An Optimized Approach for Storing and Accessing Small Files on Cloud Storage, *Journal of Network and Computer Applications*, 35(6): 1847-1862.

[8] Dong, B.; et al. (2010); A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files, *IEEE International Conference on Services Computing (SCC)*, Florida, USA: IEEE, DOI:10.1109/SCC.2010.72.

[9] Golpayegani, N.; M. Halem. (2009); Cloud Computing for Satellite Data Processing on High End Compute Clusters, *IEEE International Conference on Cloud Computing*, Bangalore, India: IEEE, 88-92, DOI:10.1109/CLOUD.2009.71.

[10] Krishna, M.; et al. (2010); Implementation and Performance Evaluation of a Hybrid Distributed System for Storing and Processing Images from the Web, *2nd IEEE International Conference on Cloud Computing Technology and Science*, Indianapolis, USA: IEEE, 762-767, DOI:10.1109/CloudCom.2010.116.

[11] Kocakulak, H.; T. T. Temizel. (2011); MapReduce: A Hadoop Solution for Ballistic Image Analysis and Recognition, *International Conference on High Performance Computing and Simulation (HPCS)*, İstanbul, Turkey, 836-842, DOI:10.1109/HPCSim.2011.5999917.

[12] http://opencv.org

[13] http://wiki.apache.org/hadoop/SequenceFile

[14] Liu, X.; et al. (2009), Implementing WebGIS on Hadoop: A Case Study of Improving Small File I/O Performance on HDFS, *IEEE International Conference on Cluster Computing and Workshops*, Louisiana USA: IEEE, 1-8, DOI:10.1109/CLUSTR.2009.5289196.

[15] https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapred/lib/ CombineFileInputFormat.html