The Design and Implementation of Image Parallel Processing Framework Based on Hadoop

Shenkuo Wang, Shaofei Wu*, Yiqi Jiang, Huajie Zhang, Ning Xia

Laboratory of Intelligent Robot, Wuhan Institute of Technology, Wuhan, China

School of Computer Science & Engineering, Wuhan Institute of Technology, 430205, Wuhan, China

*Corresponding author: wasbfc@yeah.net

Keywords: Hadoop, MapReduce, Face Detection, Image Processing Framework

Abstract: For the efficiency of image processing in traditional single-machine environment is low, and the image processing used Hadoop cluster has too large load of the main node. In this framework, the internal representation of images serialized in Hadoop and the image storage model named ImgBundleFile was designed, the OpenCV image processing library was introduced and the image processing interface using the Java development language was designed. In order to greatly shorten the image processing time of the framework, a MapTask load balancing strategy is proposed. Using framework for the face detection, experimental results show that the image parallel processing framework based on Hadoop has good stability and scalability, and the efficiency is significantly improved compared with the traditional single-machine environment.

1. Introduction

With the continuous development of information technology, network data is showing an exponential growth, so more and more research is devoted to big data and cloud computing [1]. As one of the most popular distributed computing frameworks, MapReduce is widely used for its scalability and high fault tolerance. Hadoop is an open source implementation of the Apache Foundation. It consists of three core components, including MapReduce, Hadoop Distributed File System and YARN [2].

Since the purpose of Hadoop design is to process a large amount of text data, it is still in the research stage in the field of image processing. Wu [3] have proposed using web data mining algorithm combined with MapReduce programming model to improve processing data volume and processing speed. Chang [4] have proposed using Hadoop technology and the improved Meanshift algorithm for image segmentation,

When processing high-resolution satellite images, the data processing speed is significantly improved. For medical images, there are methods already used to store data with Hadoop and HBase, mainwhile, minimize data transfer. Cao [5] have proposed using Hadoop and Otsu-Canny Operators for parallel image edge detection, which increase the detection rate.

The serialized data types currently supported by Hadoop are IntWritable, FloatWritable, BooleanWritable, etc. There is no data type that directly supports images. Because of Hadoop's NameNode records the meta information of each data, the meta information includes the number of copies of the data and the location of the stored nodes, meanwhile, it responsible for the operation of file metadata information and processing client requests, a large number of images stored in HDFS will cause the NameNode node to be overloaded, which will become the bottleneck of the Hadoop image processing framework. Hui [6] have proposed merging files into data blocks to optimize the storage of HDFS small file data and processing programs for efficient operation of the system. Zhou and Wen [7] have proposed a new Hadoop small file storage scheme based on prefetching mechanism.

In this paper, the storage model ImgBundleFile is designed for image storage in HDFS. The existing Hadoop-based image processing framework has HIPI, which combines images into a HIB file as input to MapReduce. Qureshi [8] have used Image processing with HIPI. Zhao [9] have

proposed using Hadoop Image Processing Interface (HIPI) library to extract the feature of image, experiments show that the overall speed is improved.

But HIPI is insufficient, the disadvantage is that the HIB file needs to be transcoded and decoded, and the project stops in 2016. Updates are no longer sufficient to meet actual needs. The method have been proposed using the URL to read image binary stream data processing in the Map stage, but this method is inefficient due to frequent IO read and write operations. The ImgBundleFile storage model designed by this framework is compressed and stored in the form of key and value. The key stores the name of the image, and the value stores the ByteArray of the image. Hadoop is written by Java, which has poor performance in image processing. Kune [10] in order to extend the HDFS and MapReduce interfaces, XHAMI is designed. The extended interface is presented in the form of API, implementing this interface in the entity class of image processing. In this framework, the OpenCV library [11, 12] is compiled by the Ubuntu operating system, and the image processing interface is defined in the framework, as well as its implementation class. Finally, the face detection is completed by inheriting the implementation class.

2. Hadoop Distributed Framework

The Hadoop distributed framework is an open source project under the Apache Foundation. It consists of three core components: Distributed File System (HDFS), Parallel Computing Framework (MapReduce), and Resource Scheduler (Yarn). Hadoop [13] is an implementation that integrates storage, computing, and resource scheduling.

2.1 HDFS (Hadoop Distributed File System)

As a Hadoop storage system, HDFS adopts the master-slave design and consists of a NameNode, a SecondaryNameNode and multiple DataNodes. The NameNode manages the file system, and its function is to be responsible for the user's request and the metadata information of the stored data. SecondaryNameNode is the backup node of the NameNode and is a high fault tolerance performance. DataNode is a storage node for data and is a performance of high storage. HDFS allows users to upload data files to HDFS. It is physically divided into multiple data blocks according to the size of the input file, and then stored in different DataNodes. The HDFS partition size is 128M, so it can be seen that Hadoop is for a amount of text data.

2.2 MapReduce

MapReduce is a programming model for processing and generating large data sets, its processes data by the Map and Reduce functions [14]. Before the data is input into MapReduce, it will be logically divided into multiple splits, and each splits starts a MapTask process. These MapTasks can run in parallel on one node, and the degree of parallelism depends on the machine configuration and running resources. Map function that processes a key/value pair to generate a set of intermediate key/value pairs and a reduce function that merges all intermediate values associated with the same intermediate key. The execution flow of MapReduce is shown in Figure 1.

3. The Design and Implementation of The Framework

Since Hadoop is used to process large text data, the default slice size is defined as 128M. However, for images, each image is as small as 10K and 1M, which is not suitable for direct storage. This framework adopt the mothed which small files merge into large files, then store them in HDFS, due to the different number and size of images, the combined file size will also vary greatly. If the split is too large, it will cause some nodes to be busy and some nodes to be idle. If the split is too small, it will lead to generating many MapTasks. It takes a few seconds for each MapTask to be started and added to the scheduler for scheduling, when each MapTask is executed quickly, the user-submitted task will waste too much time at startup and end. Therefore, the load balancing strategy of MapTask based on Hadoop image processing framework is proposed. That is, an appropriate MapTask parallelism is about 10-100 MapTasks per node, and preferably each

MapTask has an execution time of at least one minute [15]. The size of the logical slice of Hadoop is calculated by Eq. (1).

$$splitSize = Mathmax(minSize, Mathmin(goalSize, blockSize)) \quad (1)$$

Among them, minSize is the smallest slice, the size defaults to 1B, the goalSize default value is the size of the input file, blockSize is the physical block size of Hadoop, the size defaults to 128MB, if we want to change the size of splitSize by modifying minSize and goalSize, it will cause time loss of data transfer between nodes. In order to avoid unnecessary time loss, this framework implements load balancing of MapTask by changing the size of blockSize and modify the value of yarn.scheduler.minimum-allocation-mb and yarn.nodemanager.resource.mem-ory-mb to confirm that each node has a task to run.

The Design and Implementation of Image Parallel Processing Framework Based on Hadoop

Shenkuo Wang, Shaofei Wu*, Yiqi Jiang, Huajie Zhang, Ning Xia
Laboratory of Intelligent Robot, Wuhan Institute of Technology, Wuhan, China
School of Computer Science & Engineering, Wuhan Institute of Technology, 430205, Wuhan, China
*Corresponding author: wasbfc@yeah.net

Keywords: Hadoop, MapReduce, Face Detection, Image Processing Framework

Abstract: For the efficiency of image processing in traditional single-machine environment is low, and the image processing used Hadoop cluster has too large load of the main node. In this framework, the internal representation of images serialized in Hadoop and the image storage model named ImgBundleFile was designed, the OpenCV image processing library was introduced and the image processing interface using the Java development language was designed. In order to greatly shorten the image processing time of the framework, a MapTask load balancing strategy is proposed. Using framework for the face detection, experimental results show that the image parallel processing framework based on Hadoop has good stability and scalability, and the efficiency is significantly improved compared with the traditional single-machine environment.

1. Introduction

With the continuous development of information technology, network data is showing an exponential growth, so more and more research is devoted to big data and cloud computing [1]. As one of the most popular distributed computing frameworks, MapReduce is widely used for its scalability and high fault tolerance. Hadoop is an open source implementation of the Apache Foundation. It consists of three core components, including MapReduce, Hadoop Distributed File System and YARN [2].

Since the purpose of Hadoop design is to process a large amount of text data, it is still in the research stage in the field of image processing. Wu [3] have proposed using web data mining algorithm combined with MapReduce programming model to improve processing data volume and processing speed. Chang [4] have proposed using Hadoop technology and the improved Meanshift algorithm for image segmentation,

When processing high-resolution satellite images, the data processing speed is significantly improved. For medical images, there are methods already used to store data with Hadoop and HBase, mainwhile, minimize data transfer. Cao [5] have proposed using Hadoop and Otsu-Canny Operators for parallel image edge detection, which increase the detection rate.

The serialized data types currently supported by Hadoop are IntWritable, FloatWritable, BooleanWritable, etc. There is no data type that directly supports images. Because of Hadoop's NameNode records the meta information of each data, the meta information includes the number of copies of the data and the location of the stored nodes, meanwhile, it responsible for the operation of file metadata information and processing client requests, a large number of images stored in HDFS will cause the NameNode node to be overloaded, which will become the bottleneck of the Hadoop image processing framework. Hui [6] have proposed merging files into data blocks to optimize the storage of HDFS small file data and processing programs for efficient operation of the system. Zhou and Wen [7] have proposed a new Hadoop small file storage scheme based on prefetching mechanism.

In this paper, the storage model ImgBundleFile is designed for image storage in HDFS. The existing Hadoop-based image processing framework has HIPI, which combines images into a HIB file as input to MapReduce. Qureshi [8] have used Image processing with HIPI. Zhao [9] have proposed using Hadoop Image Processing Interface (HIPI) library to extract the feature of Image,

Fig. 1. MapReduce execution process.

3.1 The Design of Image Storage Model

The problems that caused by the large number of images stored on HDFS can be summarized as follows. 1) The NameNode needs to store metadata information of these data, which requires a large amount of storage overhead. 2) The image is too small that seek physical addresses takes more time than reading and writing. 3)When the task is submitted, an image corresponds to a split, which will cause a large number of MapTasks to be started, and MapTask will cost a lot of resources to join the scheduler for scheduling [16].

At present, different solutions are proposed for a large number of small files. 1) Using text files to record the URL of images, and then directly use text files as input of MapReduce. This method needs to read through IO stream in Map phase, and write back through IO stream after processing, which results in enormous IO overhead and very low efficiency. 2) Put forward a combination of fragmented methods, which is to logically pack multiple images into a fragm-ent. When reading data through RecordReader, it reads logical fragment, but this method only solves the problem of excessive fragment and reducing the number of MapTasks, and does not solve the problem of HDFS storing a large number of images.

This framework designs a storage container through the SequenceFile interface to store a large number of images in the container [17]. The container is stored in the form (key, value), where key stores the name of the image and value stores a binary array of images. Since the basic data types that Hadoop has implemented for serialization include BytesWritable, it can be used as a serialized and deserialized storage object in the submission of Job, as well as in Map and Reduce processing. The class diagram of the storage model is shown in Figure 2.

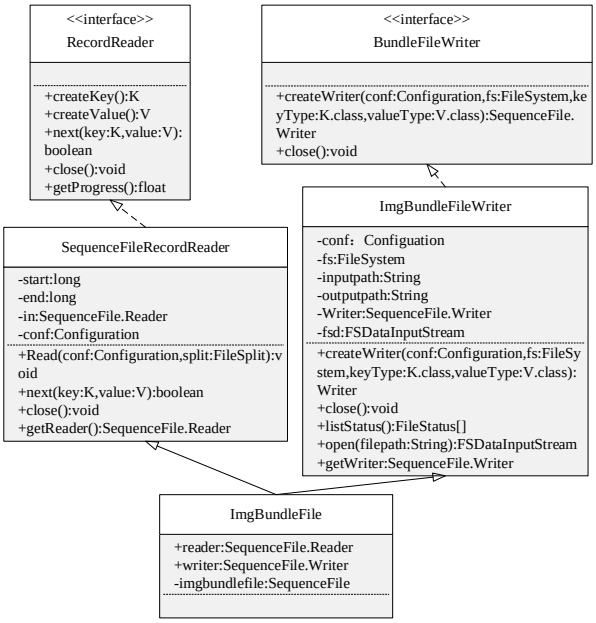

Fig. 2. Class diagram of the storage model.

3.2 The Design of Image Processing Interface

In view of the shortcomings of Java in image processing, this framework has designed an image processing interface. We only need to import the class file of the interface to use the OpenCV library function and the method of the interface.

OpenCV only provides the source package for image processing, so we need to build the corresponding C, C++ and Java compilation environment to compile the OpenCV source package. After the compilation is completed, a jar package suitable for Java and a corresponding OpenCV dynamic link library are generated. The image processing interface introduces the OpenCV dynamic link library, which realizes the conversion of Java BufferedImage data type and OpenCV Mat data type. The interface provides the output function writeImg() for local storage of data. The class diagram of the image processing interface is shown in Figure 3.

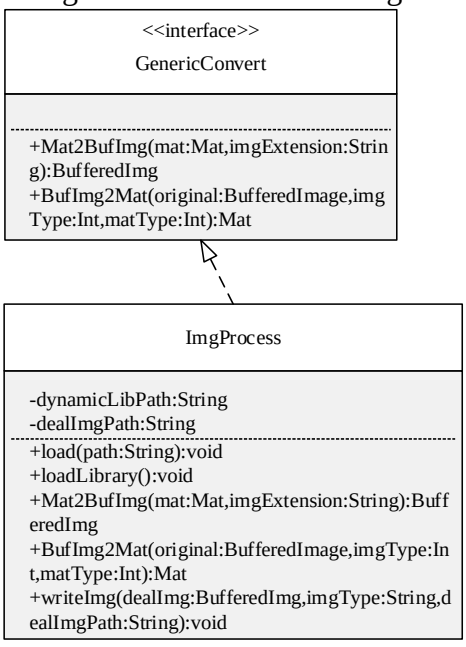

Fig. 3. Class diagram of the image processing interface.

3.3 The Design of Image Parallel Processing

According to the storage model designed in the previous section as the input of MapReduce, the input fragment size is dynamically adjusted by means of parameter transfer, and the variable

blockSize is adjusted in essence. The input fragment is read by the RecordReader and used as the input of the Map, therefore, according to the storage model, the data type of the Map function is Mapper<Text, BytesWritable, NullWritable, NullWritable>.

In the function, Text represents the type of the key and BytesWritable represents the type of the value in the storage model. The data type of the Map function is consistent with the data type of the storage model and can be serialized. Because the shuffle and reduce processes are required after the Map processing is completed, the framework does not need to go through these processes, in order to save cost and time. Therefore, the number of Reduces is set to 0, and the result of the Map is directly output and stored. The storage location is the local file system. Not HDFS.

3.4 The Face Detection Based on Adaboost

There are two main types of face detection methods: based on knowledge and based on statistics. The statistical method is to treat the face as a two-dimensional pixel matrix, construct a face pattern space through a large number of face samples, and judge whether the face exists according to the similarity. The Haar classifier is a statistically based method.

This framework uses the Haar classifier for face detection. Haar classifier consists of Haar-like feature [18], AdaBoost algorithm, cascading and integral graph fast calculation. The Haar-like feature is a predefined feature rectangle to calculate the pixel difference of the area covered by the rectangle in the original image. The commonly used Haar-like is shown in Figure 4.

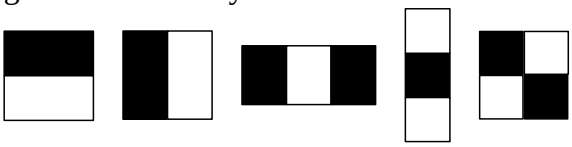

Fig. 4. Haar-like features.

Cascading is the combination of multiple strong classifiers to further process the picture, increasing the accuracy of the recognition. The integral graph function is to improve the efficiency of image feature value calculation. The value of each point in the integral graph is the sum of pixels in the upper left corner of the point. The construction of the integral graph is shown by Eq. (2).

$$SAT(x, y) = \sum_{x_i \le x, y_i \le y} I(x_i, y_i) \tag{2}$$

where SAT(i, j) represents the value of the point (x, y) on the integral map, and I(x_i, y_i) represents the pixel value of the point (x_i, y_i) in the original image.

The Adaboost algorithm generates a series of weak classifiers by training the Haar-like feature training set $M = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$, where x_i is the eigenvector, y_i is a category label with a value of "+1" or "-1". The initial weights of the samples are the same, as shown by Eq. (3).

$$D_1 = \left(w_{1,1}, w_{1,2}, \ldots, w_{1,n}\right), w_{1,i} = \frac{1}{n}, i = 1, 2, \ldots, n \tag{3}$$

In the Eq. (3), $w_{1,i}$ is the sample weight, where 1 represents the 0-th iteration and i represents the i-th sample, The sample weight will increase when misclassified by weak classifier. The weight value of the trained weak classifier is determined by its accuracy rate, and the high accuracy corresponds to the high weight of weak classifier. When the cycle is trained T times, we can get T weak classifiers. The weak classifier is shown by Eq. (4).

$$f_t(x) : \chi \to \{-1, 1\}, t = 1, 2, \ldots, T \tag{4}$$

For the weak classifier $f_t(x)$, first calculate its error rate e_t for the training samples and then calculate the weight of the current weak classifier. The formula is shown by Eq. (5) and Eq. (6).

$$e_t = \sum_{i=1}^{n} w_{t,i} I\left(f_t(x_i) \neq y_i\right) \tag{5}$$

$$\alpha_t = \frac{1}{2}\log\frac{1-e_t}{e_t} \tag{6}$$

Where α_t is the weight of the $f_t(x)$ classifier, Then update the weights of all samples $D_{t+1} = (w_{t+1,1}, w_{t+1,2}, \ldots, w_{t+1,n})$, the update formula is shown by Eq. (7) and Eq. (8).

$$w_{t+1,i} = \frac{w_{t,i}}{z_t}\exp\left(-\alpha_t y_i f_t\left(x_i\right)\right) \tag{7}$$

$$z_t = \sum_{i=1}^{n} w_{t,i}\exp\left(-\alpha_t y_i f_t\left(x_i\right)\right) \tag{8}$$

Where, Z_t is the normalization factor, which is the sum of the weights of the last iteration of all samples. $w_{t+1,i}$ is the sample weight, where $t+1$ represents the No.t iteration and i represents the i-th sample.

The linear combination of weak classifiers is constructed according to the weight of the classifier to form a strong classifier, and the strong classifier is represented by Eq. (9).

$$\mathrm{sgn}\left(\mathrm{F}\left(x\right)\right) = \mathrm{sgn}\left(\sum_{t=1}^{T}\alpha_t f_t\left(x\right)\right) \tag{9}$$

In this formula, x stands for the input vector, F(x) is a strong classifier, $f_t(x)$ is a weak classifier and α_t is the weight value of the weak classifier whose weight value is a positive number, and T is the number of weak classifiers. The output of the weak classifier is "+1" or "-1", which corresponding to the positive and negative samples, and sgn is the sign function.

Face detection on an image requires a series of weak classifiers, and the voting result of each classifier (whether the current detection area is a face) is multiplied by its classifier weight, and then summed the product results. The weighted sum results are compared with the average voting results, finally the classification results are obtained.

4. Experimental results and analysis

In this experiment, four machines are built on the vmware virtual machine, and the Hadoop cluster is fully distributed. The operating system of each node is ubuntu 14.04 LTS, and Hadoop 2.7.7 is installed and configured. The master node is the center of operation, due to the large load, so we set the number of processor cores are 2, the number of slave processor core is 1, the running memory is set to 2G, the CPU is Intel® CoreTM i5-8300H CPU @ 2.30GHz, one node is used as NameNode and ResourceManger, and the other 3 nodes are used as DataNode and NodeManger. The experimental data uses the FDDB face detection data set, which contains 2845 images taken from the Faces in the Wild data set and it has 5171 faces. The database is shown in Figure 5.

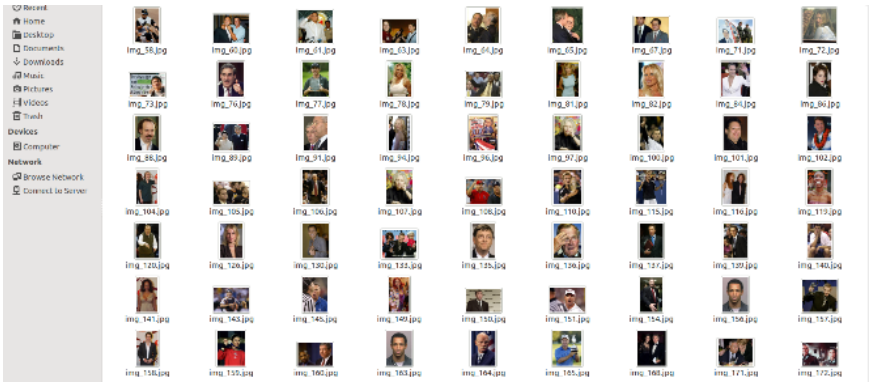

Fig. 5. FDDB DataBase.

According to the size of the data volume, this framework sets the block size to 3M and the number of copies is set to 3. Two sets of experiments were performed, one for face detection on the

Hadoop image processing framework and the other for face detection in a single-machine environment. The experiment records the time and killed times of the frame. Face detection by adding 500 images each time, at last we will record the time that the entire data set has been processed. The running results are shown in Table 1 and Table 2.

Table 1 Increase the number of images in a cluster environment.

Number of nodes	Number of images	Data size (MB)	Number of splits	Participating node	Start time	End time	killed times	Running time (s)
3	500	6.7	3	02,03,04	10:21:09	10:21:39	0	30
3	1000	13.4	5	02,03,04	10:34:11	10:35:15	1	64
3	1500	19.9	7	02,03,04	10:48:43	10:50:09	1	86
3	2000	26.6	9	02,03,04	11:01:14	11:03:11	1	117
3	2500	33.2	11	02,03,04	11:25:07	11:27:32	2	145
3	2800	37.4	13	02,03,04	11:38:21	11:40:57	3	156

Table 2 Increase the number of images in a single-machine environment.

Number of images	Start time	End time	Running time (s)
500	10:24:29	10:25:03	34
1000	10:40:10	10:41:20	70
1500	10:53:45	10:55:31	106
2000	11:05:34	11:07:59	145
2500	11:31:26	11:34:32	186
2800	11:47:36	11:51:04	208

From Table 1, it can be obtained that as the amount of data increases, the number of splits increases, and the face detection time of the cluster becomes longer, and the number of MapTask killed increases. Increasing the number of kills will result in a redistribution of tasks, and the progress of the map will be reversed, thus increasing the running time of the entire job. As can be seen from Table 2, in a single-machine environment, the running time basically increases linearly with the amount of data. By comparing Table 1 and Table 2, it can be seen that when the amount of data is small, the cluster and single machine running time are similar. When the amount of data is further increased, the processing time between the single machine and the processing time of the cluster are getting larger and larger. This indicates that when the amount of data increases gradually, the face detection time of the cluster is much smaller than that of the single machine. Therefore, the experiment proves that the face detection efficiency of this framework is better than the traditional face detection method.

In order to further study the speed of cluster processing by the number of nodes, a second experiment was done. By changing the number of cluster nodes to 1, 2, and 3, the number of face images in the photo library is from 500 to 2500. The time required for face detection is shown in Figure 6.

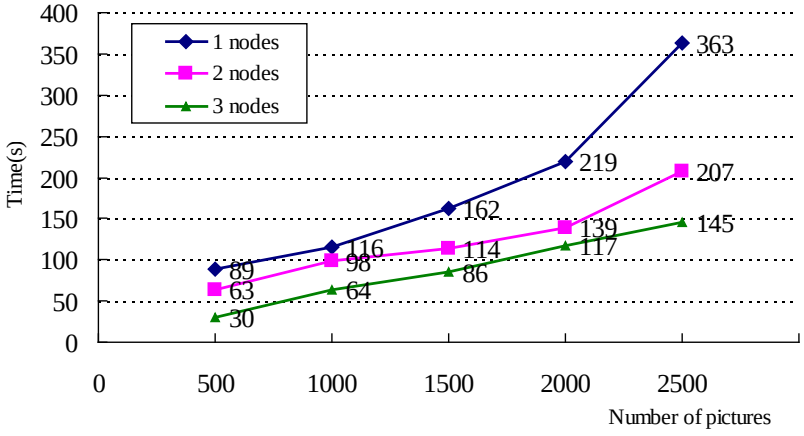

Fig. 6. Detection time of different node numbers.

It can be seen from Figure 6 that in the case where the number of images is constant, the more nodes, the shorter the face detection time, but they have less difference. As the image data continues to increase, the running time difference between different nodes is also increasing. The reason is that the number of nodes are increase, so each nodes get fewer splits, so the execution time is greatly shortened. By comparing the number of nodes to 2 and the running time of a single machine, it can be found that the running time of node number 2 is higher than that of a single machine, the reasons can be summarized as follows. 1) There are few current working nodes, so too many splits are allocated to each node. 2) Cluster startup per It takes a certain amount of time for each MapTask. If the MapTask fails to run, it will redistribute the task, which also leads to the overall running time is too long.

5. Conclusion

This framework uses the Hadoop distributed computing framework to achieve parallelization of images. The storage model is designed for the image, which saves the space overhead of the master node, designs the internal representation of the image, and implements the image processing interface. Finally, this framework is used for face detection. Experiments show that the image processing framework has the same time as a single machine when the amount of data is small and the number of running nodes is large. When the amount of data increases, the time for the frame to detect faces is significantly better than the traditional single machine mode. As the amount of data increases, by increasing the number of running nodes, the operational efficiency of the image processing framework can be increased and the stability of the cluster can be improved. And when the amount of data is large, the framework also shows good stability and efficiency. The next step will be to study how to apply face recognition to this framework to improve the efficiency of face recognition.

Acknowledgment

This work was supported by Natural Science Foundation of Hubei Province of China (Grant No.2018CFB681).

References

[1] B. Saraladevia, et al, Big Data and Hadoop-A Study in Security Perspective, Procedia Computer Science 50(2015), 596-601.

[2] Aji, Ablimit, et al, Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce, Proceedings of the VLDB Endowment 6(11) (2013), 1009-1020.

[3] Wu, Wei, Y. Chen, and D. Seng, Implementation of Web Mining Algorithm Based on Cloud Computing, Intelligent Automation & Soft Computing 23(4) (2017), 599-604.

[4] Shengpeng, Chang, et al, A kind of high resolution remote sensing image processing method based on Hadoop, Computer Engineering & Applications 51(11) (2015), 167-171.

[5] Jianfang, Cao, et al, Implementing a Parallel Image Edge Detection Algorithm Based on the Otsu-Canny Operator on the Hadoop Platform, Computational Intelligence and Neuroscience 2018(2018), 1-12.

[6] He, Hui, et al, Optimization strategy of Hadoop small file storage for big data in healthcare, The Journal of Supercomputing 72(10) (2016), 3696-3707.

[7] Zhou, Hui Xiang, and Q. Y. Wen, New Solution for Small File Storage of Hadoop Based on Prefetch Mechanism, Advanced Materials Research 981(2014), 205-208.

[8] Qureshi, Basit, et al, Performance of a Low Cost Hadoop Cluster for Image Analysis in Cloud Robotics Environment, Procedia Computer Science 82(2016), 90-98.

[9] Zhao, Y. X., Zhang, W. X., Li, D. S., Huang, Z., Li, M. N., & Lu, X. C, Pegasus: a distributed and load-balancing fingerprint identification system, Frontiers of Information Technology & Electronic Engineering 17(8) (2016), 766-780.

[10] Kune, R., Konugurthi, P. K., Agarwal, A., Chillarige, R. R., & Buyya, R, Xhami - extended hdfs and mapreduce interface for big data image processing applications in cloud computing environments, Software—practice & Experience 47(2017), 43-51.

[11] Druzhkov, P. N., et al, New object detection features in the OpenCV library, Pattern Recognition & Image Analysis 21(3) (2011), 384-386.

[12] Domínguez, C., Heras, J., & Pascual, V, Ij-opencv: combining imagej and opencv for processing images in biomedicine, Computers in Biology & Medicine 84(2017), 189-194.

[13] Rathore, M. M., Son, H., Ahmad, A., Paul, A., & Jeon, G, Real-time big data stream processing using gpu with spark over hadoop ecosystem, International Journal of Parallel Programming 46(3) (2018), 630-646.

[14] Heintz, B., Chandra, A., Sitaraman, R., & Weissman, J, End-to-end optimization for geo-distributed mapreduce, IEEE Transactions on Cloud Computing 4(3) (2017), 293-306.

[15] V. M. Arul Xavier, and S. Annadurai, Chaotic social spider algorithm for load balance aware task scheduling in cloud computing, Cluster Computing 9(2018), 1-11.

[16] Zhang, Y., Zhu, Z., Cui, H., Dong, X., & Chen, H, Small files storing and computing optimization in hadoop parallel rendering, Concurrency & Computation Practice & Experience 29(20) (2017), 1269-1274.

[17] Dyer, R., Nguyen, H. A., Rajan, H., & Nguyen, T. N, Boa:ultra-large-scale software repository and source-code mining, Acm Transactions on Software Engineering & Methodology 25(1) (2015), 1-34.

[18] Jiang, S., Ning, J., Cheng, C., & Li, Y, Robust struck tracker via color haar-like feature and selective updating, Signal Image & Video Processing 11(6) (2017), 1-8.