Improving Performance of Content-Based Image Retrieval Schemes using Hadoop MapReduce

[1]Wichian Premchaiswadi, [2]Anucha Tungkatsathan, [3]Sarayut Intarasema
Graduate School of Information Technology
Siam University
Bangkok, Thailand
[1]wichian@siam.edu, [2]aimdala@hotmail.com, [3]sarayut@siamu.ac.th

Nucharee Premchaiswadi
Faculty of Information Technology
Dhurakij Pundit University
Bangkok, Thailand
nucharee@dpu.ac.th

Abstract— **In this paper, a HadoopMapReduce framework is presented in order to perform parallel processing used for an online CBIR application. HadoopMapReduce framework is used with the intention of integrating an image analysis algorithm into the text-based image search engines without degrading their response time. Therefore, the main objective of the study is a distribution of the image data over a large number of nodes. Some of the techniques used in the paper includes: image indexing and retrieval, parallel processing of downloading images, indexing, and comparing the similarity of retrieved images from various sources.**

Keywords— HadoopMapReduce; MapReduce; Parallel Processing; Image Processing

I. INTRODUCTION

Content-Based Image Retrieval (CBIR) has been an ongoing area of research for decades but yet it is not being appeared in the mainstream image searching [1]. Most of the current CBIR systems presented in academic and research papers tend to focus on new technical issues and algorithms, and frequently ignore the end user point of the view. Some limitations of current CBIR systems were addressed by Theo Pavlidis [3].

To deal with realistic situations, we present a joint querying and relevance feedback scheme based on the both high-level and low-level features of images for an on-line content-based image retrieval system [16], [17]. Our intention is to overcome the common limitations of the existing keyword-based and content-based approached of image retrieval systems. Although the result in term of accuracy is well done but we found that a computation time based on the proposed multi-threading framework cannot deal with a time consuming task of image processing in real time systems efficiency. The main drawback is that parallel processing using multithreading cannot merely deal with a complexity of images in terms of feature extraction and feature similarity computation. Based on the experimental results in [2], a processing time to retrieve images for one query was about one minute to calculating 200 images [2]. Therefore, the search results are not satisfactory in terms of time for retrieving images. At this point, there are many researchers are trying to solve this problem by using distributed computing (i.e., for instance, cluster computing to reduce the computational time) [4] [5] [6]. Yongquan Lu, et al [4] presented a parallel technique to perform feature extraction and a similarity comparison of visual features developed on cluster architecture. The following conducted experiments showed that a parallel computing technique can be applied so as to significantly improve the performance of a retrieval system. Kao, et al [5] proposed a cluster platform which supports the implementation of retrieval approaches used in CBIR systems. Their paper introduces the basic principles of image retrieval with dynamic feature extraction using cluster platform architecture. The main focus in [5] is workload balancing across the cluster with a scheduling heuristic and execution performance measurements of the implemented prototype. As a result, cluster computing reduces the computational time. Nevertheless, to design a distributed algorithm and program it with cross-platform capability is a difficult task [10].

To overcome the drawbacks of the existing parallel computing technique using cluster platform architecture, we present a parallel processing method using Hadoop MapReduce[11] [12] [13]. In this paper, we utilize the search engine to retrieve a large number of images using a given text-based query. In the low-level image retrieval process, the system provides a similar image search function for the user to update the input query for image similarity characterization. A high-level semantic retrieval can be done by using relevance images from Yahoo image search engine. For low-level feature, we introduce a fast and robust color feature extraction technique namely auto color correlogram and correlation (ACCC) [14] algorithms with the purpose of extracting and indexing low-level features of images. Accordingly, the retrieval performance is satisfactory and higher than the

average precision of the retrieved images using autocorrelogram (AC). Moreover, It can reduce computational time from O(m2d) to O(md) [14]. The framework of Hadoop MapReduce is proposed to incorporate an image analysis algorithm into the text-based image search engines. It enhances the capability of an application when downloading images, indexing, and comparing the similarity of retrieved images from diverse sources [19], [20].

Section II proposes the framework of HadoopMapReduce for an on-line image retrieval system. Section III discusses the proposed indexing technique for image visual content. Section IV discusses map and reduce tasks. Concluding remarks are set out in Section V.

II. The Proposed Framework

A. Preliminary

Before introducing our framework of parallel processing using HadoopMapReduce for a joint querying image search scheme, we briefly examine the properties of the queries to be answered. We have developed a novel framework of real-time processing for an on-line CBIR system using relevant images from Yahoo images search engine. Our method uses the following steps: (a) Yahoo Images are first used to obtain a large number of images that are returned for a given text-based query; (b) The users can select any certain images to perform an update for the input query of the image similarity characterization; (c) A HadoopMapReduce processing method is used to manage and perform data parallelism (or loop-level parallelism such as downloading images, extraction of visual features and computation of visual similarity measures); (d) If necessary, users can also change a keyword before selecting a relevance image for the query; (e) The updated queries are further used to adaptively create a new answer for the next set of returned images according to the users' preferences [18]. The overview of our proposed image search scheme is shown in Fig. 1.

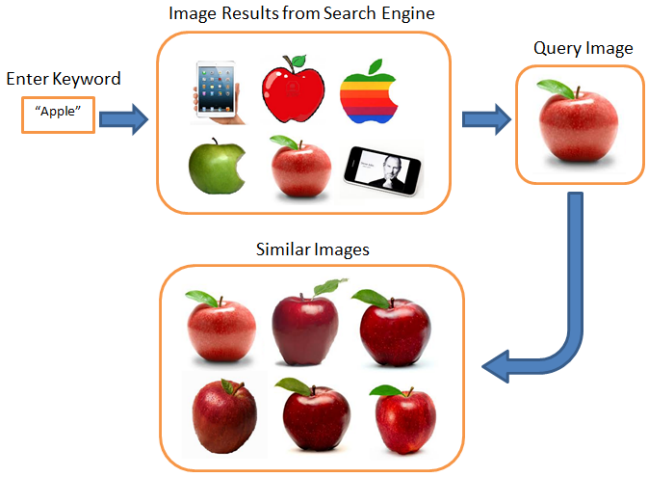

Fig. 1. Proposed image search scheme.

B. Hadoop MapReduce

Fig. 2 illustrates the MapReduce framework that consists of subcategories as follows: (1) Map Processing: HDFS splits the large input data set into smaller data blocks (64 MB by default) controlled by the property dfs.block.size,(2) Spill: When the buffer size reaches a threshold size controlled by io.sort.spill.percent (default 0.80, or 80%), a background thread starts to spill the contents to disk,(3) Partitioning: before writing to the disk the background thread divides the data into partitions,(4) Sorting: memory sort is performed on key (based on the method of key class),(5) Merging: Before the map task is finished, the spill files are merged into a single partitioned and sorted output file,(6) Compression: The map output can be compressed before writing to the disk for faster disk writing, lesser disk space, and reducing the amount of data to transfer to the reducer, (7) Reduce Operations: the reducer has three phases as following: Copy, Sort and Reduce [13].

Hadoop divides the input to a MapReduce job into fixed-size pieces called "input splits". Hadoop creates one map task for each split which runs the user-defined map function for each record in the split. Having many splits means the time taken to process each split is small compared to the time to process the whole input. So if the splits are processed in parallel, the processing is better load-balanced because the splits are small and a faster machine will be able to process proportionally more splits over the course of the job than a slower machine. Even if the machines are identical, failed processes or other jobs running concurrently make load balancing desirable, and the quality of the load balancing increases as the splits become more fine-grained. On the other hand, if splits are too small, then the overhead of managing the splits and of map task creation begins to dominate the total job execution time. For most jobs, a good split size tends to be the size of a HDFS block, 64 MB by default, although this can be changed for the cluster (for all newly created files), or specified when each file is created.

The whole data flow with a single reduce task is illustrated in Fig. 2. The dotted boxes indicate nodes, the light arrows show data transfers on a node, and the heavy arrows show data transfers between nodes. When there are multiple reducers, the map tasks partition their output while each creating one partition for each reduce task. There can be many keys and their associated values in each partition, but the records for every key are all in a single partition. The partitioning can be

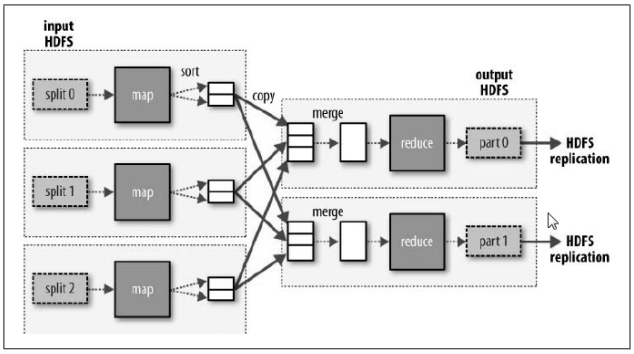

Fig. 2. Shuffle and sort in MapReduce[12].

controlled by a user-defined partitioning function, but normally the default partitioner—which buckets keys using a hash function—works very well. This diagram makes it clear why the data flow between map and reduce tasks is colloquially known as "the shuffle," as each reduce task is fed by many map tasks.

Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows theuser to specify a combiner function to be run on the map output—the combiner function's output forms the input to the reduce function. Since the combiner function is an optimization, Hadoop does not provide a guarantee of how many times it will call for a particular map output record. In other words, calling the combiner function zero, one, or many times should produce the same output from the reducer.

In general, Hadoop Distributed File System (HDFS) consists of many components and specific roles as follows: (1) NameNode: HDFS cluster consists of a single NameNode for managing the file system namespace and regulates access to file by clients, (2) DataNodes: Each slave machine in the cluster will host a DataNode daemon to perform the grunt work of the distributed file system both reading and writing, (3) SecondaryNameNode (SNN): SNN is an assistant daemon for monitoring the state of the cluster HDFS, (4) Job Tracker: The Job Tracker daemon is the liaison between the application and Hadoop, (5) TaskTracker: TaskTracker manages the execution of individual tasks on each slave node. MapReduce

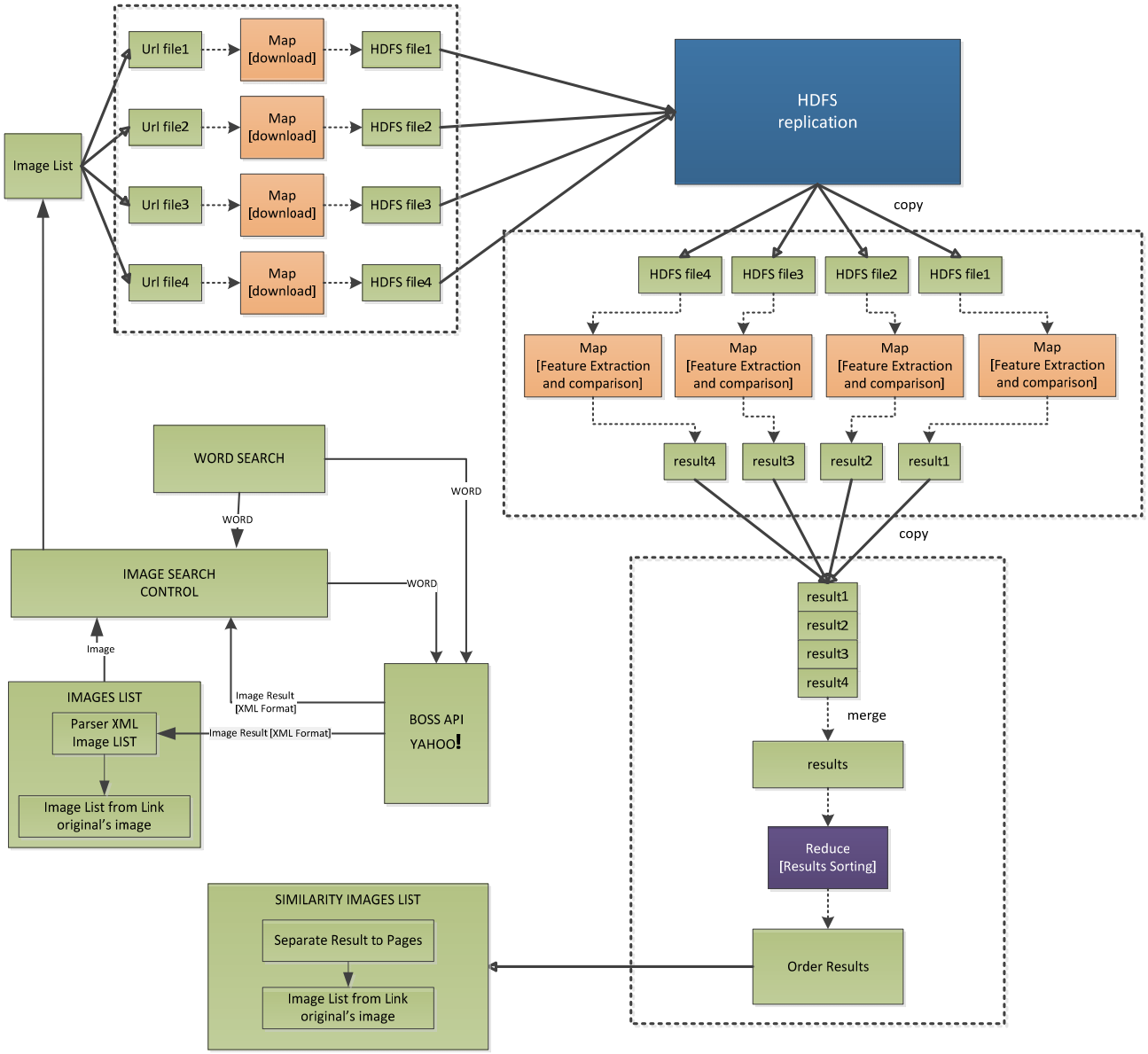

Fig. 3. A HaDoop MapReDuce Framework for Online CBIR System

programs are executed in two main phases called "mapping" and "reducing". Each phase is defined by a data processing function and these functions are called "mapper" and "reducer", respectively. In the mapping phase, MapReduce takes the input data and feeds each data element to the mapper. In the reducing phase, the reducer processes all the outputs from the mapper and arrives at a final result as shown in Fig. 2.

C. A HaDoop MapReDuce Framework for Online CBIR Systems.

In this paper, a HaDoopMapReDuce processing method is used to carry out parallel processing for a specific purpose. HaDoopMapReDuce is a programming model for data processing. To take advantage of the parallel processing that Hadoop provides, we need to express our query as a MapReduce job. After some local and small-scale testing, a MapReduce job is run on a cluster of machines. In order to utilize the MapReduce more efficiently, the number of map/reduce slots should be considered and they must technically be assigned to the correct parts of the program. The following steps explain how the MapReduce is utilised to improve the overall CBIR system performance. First, they help improve the download speed for retrieving images from various sources according to the locations specified in the xml file that are returned from Yahoo BOSS API [20] (after querying by the keyword). Second, they are used in calculating the image feature extraction. Last, they increase the speed of computing the similarity measure of feature vectors. The Hadoop Map/Reduce system accesses the image data through the 'namenode'. The MapReduce framework and the Hadoop Distributed File System (HDFS) are used to process vast amounts of image data in parallel and on large clusters. A Map/Reduce job splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. It sorts the outputs of the maps which are then input to the reduce tasks. Both the input and the output of the job are stored in a file-system (HDFS files). Fig. 3 illustrates a HaDoop MapReDuce Framework for Online CBIR Systems. A HaDoop MapReDuce Framework takes care of the scheduling tasks, monitoring them and also re-executes the failed tasks.

III. FEATURE COMPUTATION

Standard HadoopMapReduce programs handle input and output of data very effectively, but struggle in representing images in a format that is useful for researchers. Current methods involve significant overhead to obtain standard float image representation. For instance, to distribute a set of images, Map nodes would require a user to pass the images as a String, then decode each image in each map task before being able to do access pixel information. This is not only inefficient, but inconvenient. These tasks can create headaches for users and make the code look cluttered and difficult to interpret the intent of the code. Accordingly, sharing code is less efficient because the code is harder to read and harder to debug.

Our paper mainly focuses on parallel computing techniques for image retrieval. The main objective is to reduce the processing real-time in a CBIR system. However, an efficient color descriptor technique for image feature extraction is still required to reduce the processing time. In this section, we present an efficient algorithm used for image feature extraction. The algorithm is an extension of the correlogram technique for color indexing. An auto color correlation (ACC) [14] expresses how to compute the mean color of any pixels of color C_j at a distance k-*th* from a pixel of color C_i in the image. Formally, the ACC of image $\{I(x,y), x = 1,2,...,M, y = 1,2,...,N\}$ is defined as

$$ACC(i,j,k) = MC_j\gamma^k_{c_i,c_j}(I) \mid c_i \neq c_j \qquad (1)$$

Where the original image $I(x,y)$ is quantized to j colors $C_1,C_2,...,C_j$ and the distance between two pixels $k \in [\min\{M,N\}]$ is fixed a priori (e.g. 1,3,5,7,9). Let MC_j is the color mean of color C_j from color C_i at distance k in an image I. The mean colors are computed as follows:

$$r\gamma^{(k)}_{c_ic_j}(I) = \frac{\Gamma^{(k)}_{c_i,rc_j}(I)}{\Gamma^{(k)}_{c_i,c_j}(I)} \mid c_i \neq c_j \qquad (2)$$

$$g\gamma^{(k)}_{c_ic_j}(I) = \frac{\Gamma^{(k)}_{c_i,gc_j}(I)}{\Gamma^{(k)}_{c_i,c_j}(I)} \mid c_i \neq c_j \qquad (3)$$

$$b\gamma^{(k)}_{c_ic_j}(I) = \frac{\Gamma^{(k)}_{c_i,bc_j}(I)}{\Gamma^{(k)}_{c_i,c_j}(I)} \mid c_i \neq c_j \qquad (4)$$

Where denominator $\Gamma^{(k)}_{c_i,rc_j}(I)$, $\Gamma^{(k)}_{c_i,gc_j}(I)$ and $\Gamma^{(k)}_{c_i,bc_j}(I)$ are the total of pixels values (RGB Color Space) of color C_j at distance k from any pixel of color C_i and denoted $C_j \neq 0$. N is the number of accounting color C_j from color C_i at distance k, defined by:

$$N = \Gamma^{(k)}_{c_i,bc_j}(I) = \begin{Bmatrix} P(x_1,y_1) \in C_i \mid P(x_2,y_2) \in C_i; \\ k = \min\{\mid x_1 - x_2 \mid, \mid y_1 - y_2 \mid\} \end{Bmatrix} \qquad (5)$$

We propose an extended technique of ACC based on the autocorrelogram, namely Auto Color Correlogram and Correlation (ACCC). It is the integration of Autocorrelogram and Auto Color Correlation techniques[14]. However, A complexity of ACCC is still $O(md)$. The Auto Color Correlogram and Correlation is defined as

$$ACCC(i,j,k) = \gamma^k_{c_i,c_j}(I), MC_j\gamma^k_{c_i,c_j}(I) \qquad (6)$$

Hence, we can rewrite the ACCC's computational procedure

as follows.

Algorithm: Auto Color Correlogram and Correlation
For every k distance {

For every X position

For every Y position {

Ci ← current pixel

While (Cj ← Get neighbor pixel of Ci at distance K) {

For every color Cm {

If (Ci ≠ Cj){

colorCount[k][Ci]++

colorR[k][Ci] = colorR[k][Ci] + colorRCj

colorG[k][Ci] = colorG[k][Ci]+ colorGCj

colorB[k][Ci] = colorB[k][Ci] + colorBCj

$$ACC[k][C_i] := \begin{Bmatrix} colorR[k][C_i]/(colorCount[k][C_i]*k*8) \\ colorG[k][C_i]/(colorCount[k][C_i]*k*8) \\ colorB[k][C_i]/(colorCount[k][C_i]*k*8) \end{Bmatrix}$$

}else{

colorCount[k][Ci]++

color[k][Ci] = color[k][Ci] + 1

CC[k][Ci] = color[k][Ci] / (colorCount[k][Ci] * k*8)

}

}

}

}

Basically, a similarity metric is needed for image retrieval. The type of similarity measure depends on the technique used for feature extraction. This paper we use a common similarity measure. Let the ACCC pairs for the m color bin be (α_i,β_i) in I and (α'_i,β'_i) in I'. The similarity of the images is measured as the distances between the AC's and ACC's $d(I,I')$ and is applied from [15] as follows:

$$d(I,I') = \lambda_1 \sum_{\forall i} \frac{|\alpha_i - \alpha'_i|}{1+|\alpha_i + \alpha'_i|} + \lambda_2 \sum_{\forall i} \frac{|\beta_i - \beta'_i|}{1+|\beta_i + \beta'_i|} \quad (7)$$

Where $\lambda_1$and λ_2 are the similarity weighting constants of autocorrelogram and auto color correlation, respectively. In the experiments conducted, $\lambda_1 = 0.5$ and $\lambda_2 = 0.5$. α_i and β_i are defined as follows:

$$\alpha_i = \gamma^k_{c_i,c_j}(I) \quad (8)$$

$$\beta_i = \begin{Bmatrix} rc_j\gamma^k_{c_i,c_j}(I), gc_j\gamma^k_{c_i,c_j}(I), \\ bc_j\gamma^k_{c_i,c_j}(I) \mid c_i \neq c_j \end{Bmatrix} \quad (9)$$

The detail of ACC and ACCC algorithms are presented in [14].

IV. MAP AND REDUCE TASKS

Fig. 3, MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output. The input to the map phase is the raw URL data. The image files are pulled out according to the specified URLs and a feature index of each image is calculated. The map function is a data preparation phase, setting up the data in such a way that the reducer function can do its work on it: sorting the images. The map function is also a good place to drop bad records: here we filter out images that are missing or suspect. The following sample lines of input data are shown in Fig. 4. These lines are presented to the map function as the key-value pairs (an index representing each downloaded image in Fig. 5).

<ysearchresponse responsecode="200'>
<prevpage>
/ysearch/images/v1/apple?format=xml&count=1&appid=PRG2QifIkY1cA3fLzhgwIVlr48rjlVUmaiWa&start=0
</prevpage>
<nextpage>
/ysearch/images/v1/apple?format=xml&count=1&appid=PRG2QifIkY1cA3fLzhgwIVlr48rjlVUmaiWa&start=2
</nextpage>
<resultset_images count="1" start="1" totalhits="16836085" deephits="16836085">
<result>
<abstract>
Tras muchos años de disputas y acuerdos de concordia Apple Computer y Apple Corps la discografía que gestiona los derechos de Los Beatles podrían estar a punto de llegar a un acuerdo
</abstract>
<clickurl>
http://lrd.yahooapis.com/
_ylc=X3oDMTRjMHJ2aTlsBF9TAzIwMjMxNTI3MDIEYXBwaWQDUFJHMlFpZklrWTFjQTNmTHpoZ3dsVmxyN
DhyamxWVW1haVdhBGNsaWVudANib3NzBHNlcnZpY2UDQk9TUwRzbGsDdGl0bGUEc3JjcHZpZANCYm43
SjBnZUF1MTdvY3JoeGZKLndlQVpkeTQzU2tvellHVUFDZG5k/SIG=1235rt557/**http%3A//
es.appleweblog.com/wp-content/uploads/2006/11/apple.jpg
</clickurl>
<date>2006/11/27</date>
<filename>apple jpg</filename>
<format>jpeg</format>
<height>330</height>
<mimetype>image/jpeg</mimetype>
<refererclickurl>
http://lrd.yahooapis.com/
_ylc=X3oDMTRjMHJ2aTlsBF9TAzIwMjMxNTI3MDIEYXBwaWQDUFJHMlFpZklrWTFjQTNmTHpoZ3dsVmxyN
DhyamxWVW1haVdhBGNsaWVudANib3NzBHNlcnZpY2UDQk9TUwRzbGsDdGl0bGUEc3JjcHZpZANCYm43
SjBnZUF1MTdvY3JoeGZKLndlQVpkeTQzU2tvellHVUFDZG5k/SIG=12sasnpj5/**http%3A//
es.appleweblog.com/2006/11/27/apple-y-los-beatles-a-punto-de-llegar-a-un-acuerdo
</refererclickurl>
<refererurl>
http://es.appleweblog.com/2006/11/27/apple-y-los-beatles-a-punto-de-llegar-a-un-acuerdo
</refererurl>
<size>19100</size>
<thumbnail_height>102</thumbnail_height>
<thumbnail_url>http://thm-a01.yimg.com/image/2f31a32a15522fac</thumbnail_url>
<thumbnail_width>140</thumbnail_width>
<title>apple jpg</title>
<url>
http://es.appleweblog.com/wp-content/uploads/2006/11/apple.jpg
</url>
<width>450</width>
</result>
</resultset_images>
</ysearchresponse>

Fig. 4. The key-value pairs

URL1:

(86, http://es.appleweblog.com/wp-content/uploads/2006/11/apple.jpg)

…

Fig. 5. The key-value pairs

The output from the map function is processed by the MapReduce framework before being sent to the reduce function. This processing sorts and groups the key-value pairs by key. All the reduce function has to do is iterate through the list and sort all images in each HDFS file.

V. CONCLUSION

This paper presents a framework to perform parallel processing of online CBIR application as a result of applying a HadoopMapReduce processing method. We illustrated the method to apply the "HadoopMapReduce" model on a CBIR application in more detail. In addition, the ACCC algorithm was proposed in order to reduce the processing time of feature computation. In future, the proposed framework can be implemented and tested in terms of computation time, and data distribution and allocation of the number of map/reduce slot. Finally, we believe that the proposed method can greatly reduce the processing time and therefore it can be applied to other areas of image processing as well.

REFERENCES

[1] T. Anucha and P. Wichian, "On-line Content-Based Image Retrieval System using Joint Querying and Relevance Feedback Scheme," WSEAS Transaction on Computers, Issue 5, Volume 9, May 2010

[2] T. Anucha and P. Wichian, "Multi-Threading in Real-Time Image Processing for an On-Line Content-Based Image Retrieval," Lecture Notes in Electrical Engineering, Vol. 90, 2011.

[3] T. Pavlidis, "Limitations of content-based Image Retrieval," Proceeding of the ICPR, 2008.

[4] Y. Lu et al., "Study of content-based image retrieval using parallel computing technique," Proceedings of the ATIP's, pp. 186–191, 2007

[5] O. Kao et al., "Scheduling aspects for image retrieval in cluster-based image databases," Proceedings of First IEEE/ACM Cluster Computing and the Grid, pp. 329 - 336, 2001

[6] Y. Ling, Y. Ouyang, "Image Semantic Information Retrieval Based on Parallel Computing," CCCM, vol. 1, pp. 255-259, 2008.

[7] O. Kao, "Parallel and Distributed Methods for Image Retrieval with Dynamic Feature Extraction on Cluster Architectures," dexa, pp.0110, 2001.

[8] G. Pengdong et al., "Performance Comparison between Color and Spatial Segmentation for Image Retrieval and Its Parallel System Implementation," Proceeding of ISCSCT, pp. 539-543, 2008.

[9] C. Town and K. Harrison, "Large-scale Grid Computing for Content-based Image Retrieval," Proceeding of ISKO, 2009

[10] G Yuli, F Jianping, S Shinichi. A novel Approach for Filtering Junk Images from Google Search Results. Springer LNCS 4903, pp. 1 – 12, 2008.

[11] T. White, Hadoop: the definitive guide (1st ed.). Sebastopol, CA: O'Reilly Media; 2009.

[12] J. Venner, "Pro Hadoop," Springer Berlin, 2009. pp. 177–205.

[13] P. Wichian and R. Walisa, "Optimizing and Tuning MapReduce Jobs to Improve the Large-Scale Data Analysis Process," International journal of Intelligent Systems, Vol. 00, 1-16, 2012

[14] T. Anucha el al., "Spatial Color Indexing using ACC Algorithms," Proceeding of the ICT&KE, pp. 113-117, 2009.

[15] Y. H. Lee, K. H. Lee, H. Y. Ha., Senior Member, IEEE., "Spatial Color Descriptor for Image Retrieval and Video Segmentation," IEEE Trans. Multimedia, 5(3), pp. 358–367, 2003

[16] Change, E.Y., Goh, K.S., Sychay, G., Wu, G.: CBSA: content-based soft annotation for multimodal image retrieval using Bayes point machines," IEEE Transactions on Circuits and Systems for Video Technology, 13(1), Jan 2003, pp. 26–38

[17] D. Cohn, T Hofmann, "The missing link - a probabilistic model of document content and hypertext connectivity," In: Advances in Neural Information Processing Systems 13, Cambridge, MA, MIT Press , 2001.

[18] J. Dean, S. Ghemawat, "MapReduce: simplified data processing on large clusters," Proceedings of the 6th Symposium on Operating Systems Design & Implementation, Berkeley, CA, USA, USENIX Association ,2004, pp. 10–10

[19] D. Newman, A. Asuncion, P. Smyth, M. Welling, "Distributed Inference for Latent Dirichlet Allocation" In. Proc. Neural Information Processing Systems (NIPS), Cambridge, MA, MIT Press, 2008, pp. 1081–1088

[20] S. Tong, E. Chang, "Support Vector Machine active learning for image retrieval.," In.MULTIMEDIA '01, Proceedings of the ninth ACM international conference on Multimedia, New York, NY, USA, ACM, 2001, pp. 107–118