Hindawi Publishing Corporation
Advances in Multimedia
Volume 2015, Article ID 629783, 7 pages
http://dx.doi.org/10.1155/2015/629783

Research Article

Performances Evaluation of a Novel Hadoop and Spark Based System of Image Retrieval for Huge Collections

Luca Costantini and Raffaele Nicolussi

Fondazione Ugo Bordoni, Viale del Policlinico, 147 Roma, Italy

Correspondence should be addressed to Luca Costantini; lcostantini@fub.it

Received 2 October 2015; Revised 24 November 2015; Accepted 25 November 2015

Academic Editor: Athanasios V. Vasilakos

A novel system of image retrieval, based on Hadoop and Spark, is presented. Managing and extracting information from Big Data is a challenging and fundamental task. For these reasons, the system is scalable and it is designed to be able to manage small collections of images as well as huge collections of images. Hadoop and Spark are based on the MapReduce framework, but they have different characteristics. The proposed system is designed to take advantage of these two technologies. The performances of the proposed system are evaluated and analysed in terms of computational cost in order to understand in which context it could be successfully used. The experimental results show that the proposed system is efficient for both small and huge collections.

1. Introduction

Due to the dramatic growth of available images and videos and the spread of social networks, surveillance cameras, and satellite images, an important challenge is how to effectively manage the computation and storage requirements imposed by these data [1]. The trend towards many-core processors and multiprocessor systems is thwarted by the complexity in developing applications that effectively utilize them [2]. There are many methods available for retrieving images from a collection. In the medical industry, for example, recent studies have identified in CBIR (Content-Based Image Retrieval) systems a very effective technique. In these systems, input is represented by images and output consists in all the similar images contained in the database [3, 4]. A CBIR system typically operates through three steps:

(1) extraction of the features which represent images in the collection (e.g., wavelet transform and Gabor filter bank);

(2) extraction of the features which represent the query image;

(3) comparison of the query image with the images in the collection by using the features extracted.

Often these techniques have proved to be inadequate as the number of images grows. Many studies have shown that the use of MapReduce [5] techniques can, however, greatly speed up the image processing [6, 7] while Hadoop is helpful for achieving scalability [8]. These performance improvements have to be paid in terms of constraint for data access pattern, RAM sharing rigid frameworks, and algorithms major redesign. Furthermore, a set of hybridizations of the original MapReduce framework [9] can efficiently scale the indexing pipeline across a distributed cluster of machines [10–12]. With the objective to fill the gap between complicated modern architectures and new image processing algorithms this work aims to produce a high-performance image retrieval system able to hide the software complexity from researchers allowing them to focus on designing innovative image processing algorithms. The proposed system embeds just one feature to demonstrate its effectiveness, and many other features could be embedded because the system considers the features as an abstract object. The technologies used in the system are chosen to satisfy the requirements of each specific task. To evaluate the effectiveness of the proposed system its performances, in terms of computational cost, are evaluated with collections of different size. The rest of the paper is organized as follows. In Section 2 the MapReduce

framework is explained, while in Section 3 the technologies used in the proposed system are described. The architecture of the system is described in Section 4. In Section 5 the experimental results are reported, while the conclusions are drawn in Section 6.

2. MapReduce

The MapReduce framework has been introduced by Google [13] in order to allow a distributed processing over a server cluster. Despite the traditional distributed processing framework, where the data are pushed to the nodes that belong to the cluster, which are responsible for the processing, in the MapReduce system the approach is different [10]. In this case, the data are distributed among the nodes and the tasks are pushed to the particular node that stores the data. The MapReduce framework is made up of two steps: Map and Reduce, and the whole framework is based on the concept of key, value pair $\langle k, v\rangle$ [14]. The Map function, known also as mapper, receives $\langle k^{m,\text{in}}, v^{m,\text{in}}\rangle$ as input and produces a list of pairs as output:

$$\left\langle k_1^{m,\text{out}}, v_1^{m,\text{out}}\right\rangle, \left\langle k_1^{m,\text{out}}, v_{(M-1)}^{m,\text{out}}\right\rangle, \ldots, \left\langle k_N^{m,\text{out}}, v_2^{m,\text{out}}\right\rangle, \left\langle k_N^{m,\text{out}}, v_M^{m,\text{out}}\right\rangle. \tag{1}$$

Since the key emitted from the mapper could not be unique, the Reduce function, called reducer, groups the values altogether for each unique key:

$$\left\langle k_1^{m,\text{out}}, \left[v_1^{m,\text{out}}, \ldots, v_{(M-1)}^{m,\text{out}}\right]\right\rangle \longrightarrow \left\langle k_1^{r,\text{out}}, v_1^{r,\text{out}}\right\rangle. \tag{2}$$

Depending on the implementation of the MapReduce framework, the reducer could also produce multiple key, value pairs as output. There are many advantages in using the MapReduce framework related to both data storing and processing aspects. Indeed, the file system blocks are duplicated across the nodes, ensuring in this way a great tolerance in case of node failure. Furthermore, the framework manages the block in order to minimize the traffic of network data. Regarding performances, the framework assigns the task to the nodes that are not busy, balancing in this way the load among the nodes. Finally, the framework is scalable and the number of nodes in the cluster depends only on the specific case of use. In the context of image retrieval, the MapReduce framework could be employed principally in two ways: single image processing [15, 16] or image collections processing [5]. Although it is even possible to combine these two approaches, in the proposed architecture the second approach has been adopted.

3. Technologies

The proposed system is based mainly on two technologies, Hadoop and Spark, which are described in this section. Although these technologies are based on the MapReduce framework, for many aspects they are very different and both of them are used in the proposed system. In order to take advantage of their potentialities, these two technologies are used in different parts of the system. Hadoop and Spark are described in Sections 3.1 and 3.2, respectively.

3.1. Apache: Hadoop. Hadoop, developed by the Apache Software Foundation, is an open source framework created by Doug Cutting and Mike Cafarella in 2005 [18]. Its aim is to make available a framework for distributed storage and for distributed processing. The main modules that made up the Hadoop framework are as follows:

(1) *Hadoop Common*: this module contains the libraries and the utilities.

(2) *Hadoop Distributed File System (HDFS)*: originally it was the Google File System. This module is a distributed file system used as distributed storage for the data; furthermore, it provides an access to the data with high throughput.

(3) *Hadoop YARN (MRv2)*: this module is responsible for the job scheduling and for managing the cluster resources.

(4) *Hadoop MapReduce*: originally Google's MapReduce, this module is a system, based on YARN, for the parallel processing of the data.

There are many projects related to Hadoop, such as Mahout, Hive, Hbase, and Spark. One of the main aspects that characterize Hadoop is that the HDFS has a high fault-tolerance to the hardware failure. Indeed, it is able to automatically handle and resolve these events. Furthermore, HDFS is able, by the interaction among the nodes belonging to the cluster, to manage the data, for instance, to rebalance them [19, 20]. The processing of the data stored on the HDFS is performed by the MapReduce framework. Although Hadoop is written principally in Java and C languages, it is accessible by many other programming languages. The MapReduce framework allows splitting on the nodes belonging to the cluster the tasks that have to be completed. The main drawback of Hadoop is the lack of performing efficiently real-time tasks. However, this is not an important limitation because for these specific aspects other technologies could be employed.

3.2. Apache: Spark. As well as Hadoop, Spark is a project of the Apache Software Foundation, originally created by the AMPLab at the University of California, Berkeley. Concerning the performances, the main advantage of Spark is that it outperforms Hadoop; indeed it is 100x faster for in-memory operation and 10x faster for on-disk operations. Spark adopts the MapReduce paradigm, and it is accessible by using the API by different programming languages (such as Scala, Java, and Python). The core of the system is made up of a set of powerful libraries that currently include parkSQL, Spark Streaming, MLlib, and GraphX. Spark is divided into various independent layers each one with specific responsibilities:

(1) *Scala interpreter*: it takes care of creating an operator graph through RDD (i.e., responsibility-driven design) and applying operators.

(2) *DAG scheduler*: it divides operator graph into stages. Every stage is comprised of tasks based on partitions

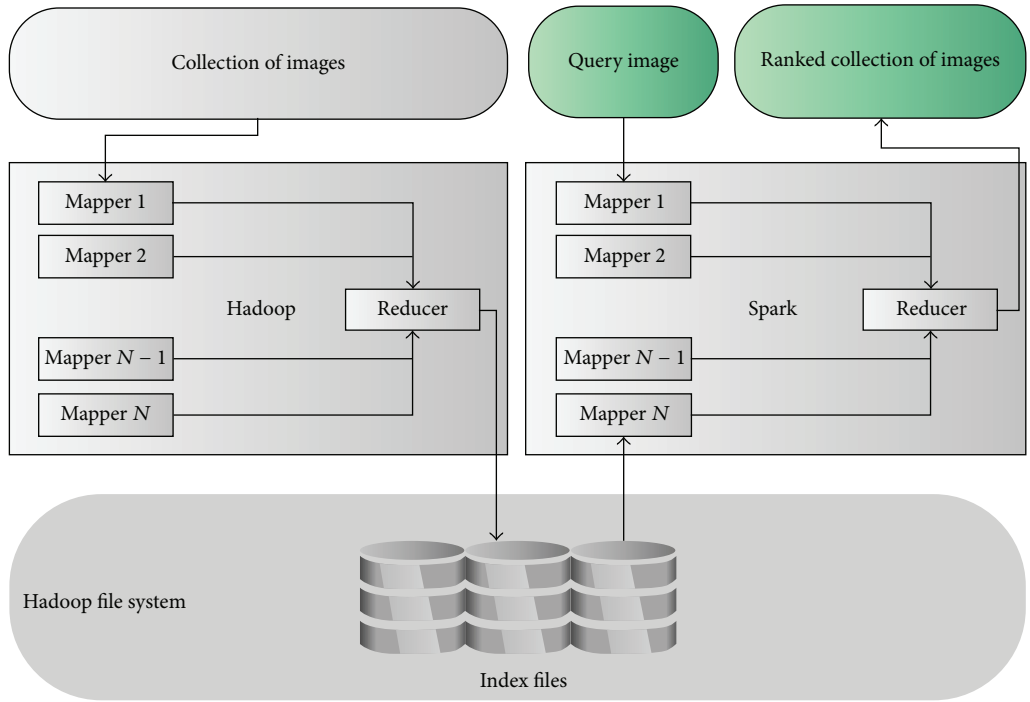

Figure 1: Architecture of the system. The gray objects represent the server side, while the green objects represent the client side.

of the input data. The DAG scheduler pipelines operators work to optimize the graph.

(3) *Task scheduler*: it activates tasks via cluster manager.

(4) *Worker*: it executes the task on a slave (i.e., the machine on which the Executor program runs).

Spark can run on the Hadoop YARN cluster and it is able to access the HDFS; this allows an easy and efficient integration of Hadoop and Spark within the same system.

4. Proposed Architecture System

The aim of this paper is to propose the architecture of a system prototype able to manage huge collection of images and deeply analyse its performances in terms of computational cost, and not in terms of recall, precision, retrieval rate, and so on, because these aspects are not important in this context. The proposed system is able to manage huge collection of images thanks to its scalability. It is based on the use of two important technologies in the context of Big Data: Hadoop and Spark; both of them employ the MapReduce framework. In the previous sections, the MapReduce framework (Section 2) as well as Hadoop (Section 3.1) and Spark (Section 3.2) is described.

Roughly speaking, the system is made up of a client side and a server side (see Figure 1). The client side allows the interactions between the users and the system by a web page. Indeed, it is used by the users to provide the query image and by the system to show the results. The processing and all the operations with a high computational cost are performed on the server side. For this reason it needs to be optimized and efficiently designed in order to manage huge collections of images. The architecture of the system, taking into account only the server side, could be considered to be made up of two main parts: one used during the indexing phase and the other one dedicated to the retrieval phase, as shown in Figure 1. However, both of them are based on a layer that is the Hadoop file system. The indexing phase and the retrieval phase have different requirements and then are separately designed. The indexing phase needs to write the index files to the file system and also has to process all the images belonging to the collection. Hadoop succeeds in an optimal way to meet these needs. On the other hand, the retrieval phase should be as fast as possible; then the operations should be done in memory; for these reasons Spark is better than Hadoop for this task. Combining these two technologies into the system it is possible to take advantage of their positive sides, minimizing the impact on the performances of their negative sides. Concerning the first part (i.e., the indexing phase), it makes use of the Hadoop technology and it stores the result (i.e., the index files) in the Hadoop file system. Based on the HIPI project [5], a list of images is assigned to each mapper and the result of the processing activity is index files with the features extracted from the images [21]. Once all the images have been processed, the reducer merges the index files created by the mappers and generates the final index file. Obviously, the computational cost, expressed in time, of this operation is

$$t^{\mathrm{I}} = t^{\mathrm{I}}_{\mathrm{sys}} + t_{\mathrm{ip}} = t^{\mathrm{I}}_{\mathrm{sys}} + \sum_{n=1}^{N} \tau_{\mathrm{ip}}(n), \tag{3}$$

where

(i) t^{I} is the total time spent for the indexing phase;

(ii) t_{ip} is the time related only to the processing of the images belonging to the collection; this is made up of the time τ_{ip} spent from each of the N mappers;

(iii) $t^{\mathrm{I}}_{\mathrm{sys}}$ is the time spent from the system, strictly related to the proposed architecture (e.g., managing of the mappers and reducer and the reading/writing operations on the Hadoop file system).

In this paper we focus our attention on $t^{\mathrm{I}}_{\mathrm{sys}}$. The second part of the system (see Figure 1) responsible for the retrieval phase is based on the Spark technology, but it makes use of the Hadoop file system to read the index file. The system performs the retrieval operation by using an image as query. From the index file the information needed to perform the comparison with the query image is loaded into memory and to each mapper is assigned a list of images, with their features, to compute the similarity with the query image. At the end of these tasks, the reducer computes the final ranking based on the similarity computed between the query image and the images within the index file. As well as for the previous part of the system, the computational cost, expressed in time, is

$$t^{\mathrm{R}} = t^{\mathrm{R}}_{\mathrm{sys}} + t_{\mathrm{ic}} = t^{\mathrm{R}}_{\mathrm{sys}} + \sum_{n=1}^{N} \tau_{\mathrm{ic}}(n), \tag{4}$$

where

(i) t^{R} is the total time spent for the retrieval phase;

(ii) t_{ic} is the time related only to compare the query image with all the images of the collection; this is made up of the time τ_{ic} spent from each of the N mappers;

(iii) $t^{\mathrm{R}}_{\mathrm{sys}}$ is the time spent from the system, strictly related to the proposed architecture (e.g., managing of the mappers and reducer, reading the index file on the Hadoop file system, performing the ranking, and processing the query image).

It is important to analyse this last component of the computational cost. Finally, in this context the color feature and its comparison metric, described and used in [22, 23], have been considered. This is not a limitation because any features can be deployed in the system, adding them to the same index file or creating a new index file for each of them, depending on the specific case of use. Obviously, when the ranking of the images belonging to the collection is based on more features, storing them into the same index file is more efficient. Otherwise, if the ranking is based on one feature, the system is more efficient when the index file stores only that feature. Furthermore, the computational cost of the processing of each image depends on which features are employed, but it does not affect the performances evaluation object of this paper. Indeed, in this paper, the computational cost introduced by the system is analysed.

5. Experimental Results

In Section 4 the architecture of the system is described; furthermore the aspects that are analysed in this paper are highlighted. As said in the previous section, the aim of this paper is to investigate the computational cost introduced by the system in both parts, the first one based on Hadoop, related to the indexing operations, and the second one based on Spark, related to the retrieval operations. In Section 5.1 the experiments performed are deeply described in order to explain the performances that are reported in Section 5.2.

5.1. Context of the Experiments. Although the technologies used by the system, described in Section 3, are scalable and are designed to be used on a cluster of machines, the experiments are not conducted on a cluster but on a single machine. Our aim is to analyse the computation cost introduced by the proposed system in order to understand its limitations in an image retrieval scenario. Our analysis is not focused on the technologies employed or on the overall performances of the system in terms of efficiency or retrieval rate; indeed just the color feature is considered. The analysis is centred on the complexity introduced by the proposed system in a context, heavy from the computational cost, such as those of image retrieval. The operations related to the indexing phase and related to the retrieval phase are performed by using different collections with a growing size. In particular, collections made up of 100, 1 K, 10 K, 100 K, and 500 K images are used. In the next section the results and the performances are extensively described.

5.2. Performances. The results of the performances analysis of the proposed system are presented in Tables 1 and 2. In particular, the results regarding the indexing phase, that is based on Hadoop, are presented in Table 1 and those regarding the retrieval phase, that is based on Spark, are presented in Table 2. Our aim is to investigate and to understand the behaviour of the proposed system for collections with different size. In our analysis the computational costs, strictly related to the feature extraction and feature comparison, are isolated. These aspects affect the performances in terms of retrieval rate, precision, and recall, but not in terms of computational cost that is analysed in this work. The feature extraction algorithm and the feature comparison algorithm are relatively simple and then their computational cost is low. This feature has been chosen for this reason because this is the worst case of use for the system. Roughly speaking, if the proposed system is efficient in this case, employing it when the features are more complex, in other words with a higher computational cost, is even more convenient.

Table 1 shows that the computational cost due to the architecture of the system becomes relevant when the collection is made up of 500 K images. This is also highlighted in the graph shown in Figure 2. Since adding nodes to the cluster impacts only on the image processing time, this aspect has to be faced when the collection size is 500 K or higher. It is important to notice that the percentage of architecture time is related to the time to process an image that, as said before, is very small for the chosen feature. In any case, a possible solution is to split the collection in order to minimize the impact of the architecture time.

Concerning the retrieval phase, the percentage of architecture time is comparable for any collection size, as shown

Table 1: Indexing phase performances (i.e., Hadoop based).

Number of images	Indexing time $[s]$, t^{I}	Time to process all images $[s]$, t_{ip}	Architecture time $[s]$, t^{I}_{sys}	Percentage of architecture time
100	29	28	1	3.45%
1000	285	280	5	1.75%
10000	2820	2800	20	0.71%
100000	28297	28000	297	1.05%
500000	165661	140000	25661	15.49%

Table 2: Retrieval phase performances (i.e., Spark based).

Number of images	Retrieval time $[s]$, t^{R}	Time to process all images $[s]$, t_{ic}	Architecture time $[s]$, t^{R}_{sys}	Percentage of architecture time
100	0.559	0.245	0.314	56.17%
1000	0.625	0.262	0.363	58.08%
10000	1.140	0.367	0.773	67.81%
100000	4.603	1.472	3.131	68.02%
500000	22.635	9.230	13.405	59.22%

Figure 2: Percentage of architecture time with respect to the number of images, indexing phase.

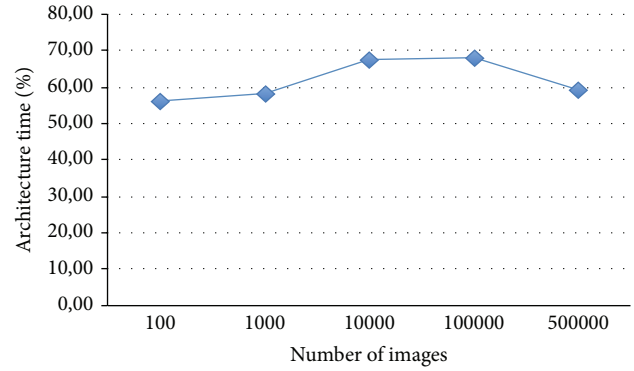

Figure 3: Percentage of architecture time with respect to the number of images, retrieval phase.

in the graph in Figure 3. This is essentially related to the time spent to process the query image and to read the index file from the Hadoop file system and write it into the memory. Although a more efficient implementation of this operation could be developed, this is a necessary operation in order to take advantage of the Spark technology. For instance, the index file could be loaded into memory only once when the application is started or when the comparison method is set. The use of this technology allows adding nodes to the cluster in order to reduce the time needed to make the comparison among the query image and the images in the index file. This aspect becomes fundamental when the comparison algorithms are complex in order to maximise the retrieval rate performances. Summarizing, the results show that, for both phases, the proposed system could be efficiently employed with small collections (e.g., ~100 images) as well as huge collections (e.g., ~500 K images), even if the chosen feature has a very low computational cost. Obviously, the impact of the architecture time decreases when more complex, and efficient, features are employed to represent the images in the collection.

In Figure 4 the ratio between the indexing time and the retrieval time, with respect to the collection size, is shown. This is an important aspect of the image retrieval system because it should be as high as possible. Indeed, since the indexing time grows as well as the collection size grows, the retrieval time should be bounded in order to not annoy the users. Furthermore, while the indexing phase is performed just once, the retrieval phase is performed much more times. This ratio, for the proposed system, has a value higher than the value obtained by the system, which is based only on Hadoop, proposed in [17]. Figure 5 refers to a collection made up of 160 K images, and the maximum value of the ratio is when the cluster is made up of 30 nodes. Also, in this case, the ratio is lower than the ratio of the proposed system for the same collection size, as shown in Figure 4, although the proposed system has been tested on a single machine; this means that the ratio could increase if the nodes are added to

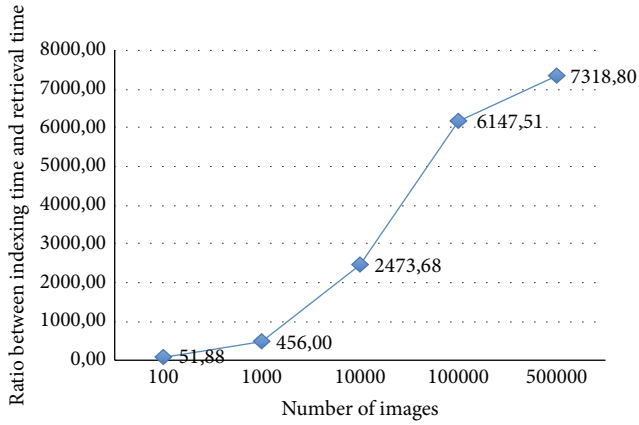

Figure 4: Ratio between indexing time and retrieval time with respect to the collection size for the proposed system.

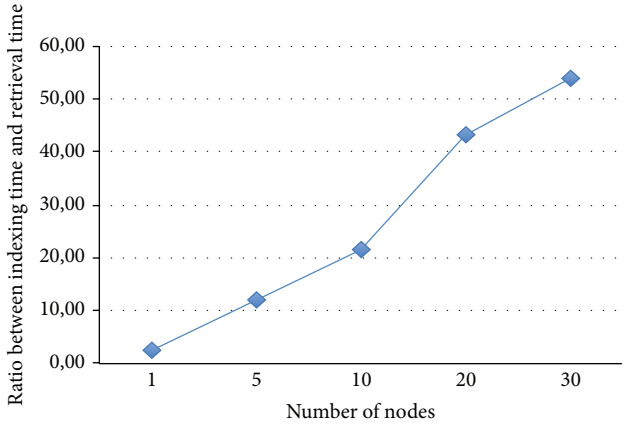

Figure 5: Ratio between indexing time and retrieval time with respect to the number of nodes for the system proposed in [17], with a collection made up of 160 K images.

the cluster. This shows that combining Hadoop and Spark in a system of image retrieval, as done in the proposed system, improves the efficiency of the whole system.

6. Conclusions

In this work a system to manage huge collection of images, based on the MapReduce framework, has been presented. The technologies used, Hadoop and Spark, allow the system to be completely scalable and to reduce the computational cost proportionally to the nodes in the cluster. Hadoop has been used in the indexing phase, while Spark has been used in the retrieval phase. The performances, in terms of computational cost, and not in terms of retrieval rate, are evaluated. A relatively cheap feature in terms of computational cost, regarding the algorithms of extraction and of comparison, has been employed in order to analyse the worst case. Furthermore, the performances have been evaluated by using collections made up of 100, 1 K, 10 K, 100 K, and 500 K images. Concerning the indexing phase, the results show that the percentage of architecture time is very low with exception for the collection made up of 500 K images. This is not a limitation because this percentage decreases when a more complex feature is used; furthermore it could be decreased by splitting the collection into smaller subcollections. On the other hand, regarding the retrieval phase, the percentage of architecture time is quite constant for all the collections but could be decreased with more efficient implementations. Future work should be focused on improving these two critical aspects and on testing the system behaviour on a cluster with a variable number of nodes. Finally, the performances show that the system is efficient for small collections (e.g., ~100 images) as well as huge collections (e.g., ~500 K images), even with one simple feature. Furthermore, the results show that the efficiency of the proposed system, based on the combination of two technologies (i.e., Hadoop and Spark), is higher than the efficiency of a system based only on Hadoop.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, "Big data analytics: a survey," *Journal of Big Data*, vol. 2, no. 1, 2015.

[2] B. White, T. Yeh, J. Lin, and L. Davis, "Web-scale computer vision using mapreduce for multimedia data mining," in *Proceedings of the 10th International Workshop on Multimedia Data Mining (MDMKDD '10)*, pp. 9:1–9:10, ACM, Washington, DC, USA, July 2010.

[3] S. Jai-Andaloussi, A. Elabdouli, A. Chaffai, N. Madrane, and A. Sekkaki, "Medical content based image retrieval by using the Hadoop framework," in *Proceedings of the 20th International Conference on Telecommunications (ICT '13)*, pp. 1–5, IEEE, Casablanca, Morocco, May 2013.

[4] S. Fong, R. Wong, and A. Vasilakos, "Accelerated pso swarm search feature selection for data stream mining big data," *IEEE Transactions on Services Computing*, 2015.

[5] S. Arietta, J. Lawrence, L. Liu, and C. Sweeney, Hipi-hadoop image processing interface, http://hipi.cs.virginia.edu/about.html.

[6] D. Moise, D. Shestakov, G. Gudmundsson, and L. Amsaleg, "Indexing and searching 100M images with map-reduce," in *Proceedings of the 3rd ACM International Conference on Multimedia Retrieval (ICMR '13)*, pp. 17–24, Dallas, Tex, USA, April 2013.

[7] L. Zhou, N. Xiong, L. Shu, A. Vasilakos, and S.-S. Yeo, "Context-aware middleware for multimedia services in heterogeneous networks," *IEEE Intelligent Systems*, vol. 25, no. 2, pp. 40–47, 2010.

[8] Y. Yan and L. Huang, "Large-scale image processing research cloud," in *Proceedings of the 5th International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING '14)*, Venice, Italy, May 2014.

[9] M. H. Almeer, "Cloud hadoop map reduce for remote sensing image analysis," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, no. 4, pp. 637–644, 2012.

[10] J. S. Hare, S. Samangooei, and P. H. Lewis, "Practical scalable image analysis and indexing using Hadoop," *Multimedia Tools and Applications*, vol. 71, no. 3, pp. 1215–1248, 2014.

[11] D. Dahiphale, R. Karve, A. V. Vasilakos et al., "An advanced mapreduce: cloud mapreduce, enhancements and applications," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 101–115, 2014.

[12] A. V. Vasilakos, Z. Li, G. Simon, and W. You, "Information centric network: research challenges and opportunities," *Journal of Network and Computer Applications*, vol. 52, pp. 1–10, 2015.

[13] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[14] N. K. Alham, M. Li, Y. Liu, and S. Hammoud, "A MapReduce-based distributed SVM algorithm for automatic image annotation," *Computers & Mathematics with Applications*, vol. 62, no. 7, pp. 2801–2811, 2011.

[15] M. Yamamoto and K. Kaneko, "Parallel image database processing with mapreduce and performance evaluation in pseudo distribuited mode," *International Journal of Electronic Commerce Studies*, vol. 3, no. 2, 2012.

[16] S. M. Banaei and H. K. Moghaddam, "Hadoop and its role in modern image processing," *Open Journal of Marine Science*, vol. 4, no. 4, pp. 239–245, 2014.

[17] D. Yin and D. Liu, "Content-based image retrial based on Hadoop," *Mathematical Problems in Engineering*, vol. 2013, Article ID 684615, 7 pages, 2013.

[18] H. Karau, *Learning Spark: Lightning-Fast Big Data Analysis*, O'Reilly Media, Sebastopol, Calif, USA, 2015.

[19] A. Shenoy, *Hadoop Explained*, Packt Publishing, 2014.

[20] T. White, *Hadoop: The Definitive Guide*, O'Reilly, 2012.

[21] H. Kocakulak and T. T. Temizel, "A hadoop solution for ballistic image analysis and recognition," in *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS '11)*, pp. 836–842, IEEE, Istanbul, Turkey, July 2011.

[22] L. Costantini, P. Sitá, L. Capodiferro, and A. Neri, "Laguerre gauss analysis for image retrieval based on color texture," in *Wavelet Applications in Industrial Processing VII*, vol. 7535 of *Proceedings of SPIE*, San Jose, Calif, USA, 2010.

[23] L. Capodiferro, L. Costantini, F. Mangiatordi, and E. Pallotti, "SVM for historical sport video classification," in *Proceedings of the 5th International Symposium on Communications Control and Signal Processing (ISCCSP '12)*, pp. 1–4, Rome, Italy, May 2012.