International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:2, 2015

A System for Analyzing and Eliciting Public Grievances Using Cache Enabled Big Data

P. Kaladevi, N. Giridharan

***Abstract*—**The system for analyzing and eliciting public grievances serves its main purpose to receive and process all sorts of complaints from the public and respond to users. Due to the more number of complaint data becomes big data which is difficult to store and process. The proposed system uses HDFS to store the big data and uses MapReduce to process the big data. The concept of cache was applied in the system to provide immediate response and timely action using big data analytics. Cache enabled big data increases the response time of the system. The unstructured data provided by the users are efficiently handled through map reduce algorithm. The processing of complaints takes place in the order of the hierarchy of the authority. The drawbacks of the traditional database system used in the existing system are set forth by our system by using Cache enabled Hadoop Distributed File System. MapReduce framework codes have the possible to leak the sensitive data through computation process. We propose a system that add noise to the output of the reduce phase to avoid signaling the presence of sensitive data. If the complaints are not processed in the ample time, then automatically it is forwarded to the higher authority. Hence it ensures assurance in processing. A copy of the filed complaint is sent as a digitally signed PDF document to the user mail id which serves as a proof. The system report serves to be an essential data while making important decisions based on legislation.

***Keywords*—**Big Data, Hadoop, HDFS, Caching, MapReduce, web personalization, e-governance.

I. Introduction

BIG data is a popular term used to describe the exponential growth and availability of data, both structured and unstructured. Knowing the people better will allow serving them better. Satisfying the need of the people is the main motto of this system by analyzing and eliciting public grievances using cache enabled bid data. In this system more data leads to difficulty in handling of data and to analyze them, so the data are handled by HDFS and MapReduce framework with caching technique [6]. MapReduce provides simple programming interface and excellent performance. Application that takes a large amount of data as input are referred to big data applications.

In MapReduce input data is split into pieces and served to the workers node in mapping phase [2]. MapReduce system parses the input splits to each worker. The intermediate results generated by worker nodes during mapping phase are shuffled and sorted by MapReduce system are served to workers node in reducing phase. Final results are generated by reducers.

Hadoop is an open-source implementation of Google MapReduce Programming model. Hadoop Distributed File System (HDFS) provides distributed file storage and is optimized for large immutable blobs of data [9]. A simple Hadoop cluster contains single master and multiple worker nodes. In this paper we present a cache description scheme that will improve the response time of system. This scheme recognizes the source input from which the cache content is generated and operations are performed on the input. We also present the additional reducer that will generate the final result from the cache node and data node. We implement the cache enable big-data by extending the components of Hadoop project.

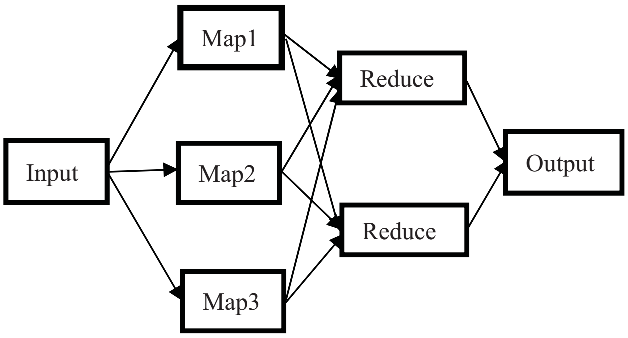

Fig. 1 Illustration on the MapReduce Programming Model

A demand of discovering knowledge from big data using statistical analysis and data mining become higher [3]. MapReduce computation comes with many security problems. Besides communication attacks such as replay, Denial of Service (DoS), MapReduce also have a privacy issue. A careless or malicious MapReduce application may expose sensitive data by writing it into a word readable file or by providing a specific result that will signal the presence of a sensitive item in datasets [11].

II. CACHE DESCRIPTION

Cache refers to the immediate data that is used to generate the partial output before processing the entire dataset. Big data Cache refers to the data which is most recently used by the MapReduce task. In this system we refer the recently submitted complaint files as the cache data. These data are stored in a Hadoop Distributed File System (HDFS). Cached data are managed in the clusters which are dedicated to store the cache data. For example, in the word count application, each mapper produces a list of {word, count} tuple that contains the count of each word in the file which are used as

P. Kaladevi and N. Giridharan are with Department of Computer Science and Engineering, Assistant Professor, from K.S.Rangasamy College Of Technology, Tiruchengode – 637215, Namakkal, Tamil Nadu, India (e-mail: kaladevi@ksrct.ac.in, giri.susee@live.com).

input to the mapper [8]. Reducer sorts and shuffles those tuples and generates the final result. We present cache cluster with two nodes one that stores recently submitted complaint files and another node that stores recently accessed complaint files. To avoid duplication the files are moved between the cache nodes and data nodes.

A. Map Phase Cache Description

In this system two types of map phase are introduced namely cache map phase and data map phase. Cache map phase splits the input from cached data to each cache worker nodes and produces records. Data map phase splits the data from data node and splits to worker nodes. Each phase performs the same task but source of the data differs [12].

The exact format of the cache description differs with respect to the specific semantic context of different application. This could be designed and implemented by application developers who are responsible for implementing MapReduce tasks.

B. Reduce Phase Cache Description

The input for the reduce phase is a list of records from the map phase. Reduce phase generates the final result by sorting and shuffling the records from the map phase. We present three types of reducer phase namely cache reduce phase, data reduce phase and final reduce phase. Cache reduce phase performs the sorting and shuffling in the data generated by cache map phase [7]. Data reduce phase performs the same operation in the data generated by data map phase. Final reduce phase produces output by computing the reducing function over the data generated by the cache reduce phase and the data reduce phase.

For example, two data files “file1.data” and “file2.data” from cache map phase are shuffled to produce two input files “input1.data” and “input2.data” for cache reducers and two data files “file3.data” and “file4.data” from data map phase are shuffled to produce two input files “input3.data” and “input4.data” for data reducers. Finally cache reducer output file “output1.data” and data reducer output file “output2.data” are shuffled to produce input files “finput1.data” and “finput2.data” for final reducer.

III. Proposed System

A. Complaint Registration

In this paper, we present a system that will allow public users to register their complaints and to track them using the web based interface. The complaints are registered through the Public grievances website. The registered complaints are stored in a Cache enabled HDFS. Once the complaint is stored, a unique complaint ID is generated for each complaint that serves as the primary key for the data stored in Hadoop. Based on the type of the complaint registered, they are categorized and processed. With the help of complaint ID, the status of the complaint can be viewed. If a complaint is not being updated regularly, it will be automatically forwarded to the higher authority for timely action and to intimate about the status of the complaints to the people.

B. Cache Item Submission

The complaints that are received from the users are stored in the Hadoop Distributed File System (HDFS). These files are not directly fed into the data node. Initially these files are stored in the node which is dedicated to store the cache files. The cache manager records the description and the file name of the cache item in the DFS. The cache item put on the different machine which contains its own mapping and reducing process [4]. Cache master continuously monitors the MapReduce task when the new files queried from the data node for the process then cache master moves that file from the data node to the cache node. Cache master moves the files instead of copying to avoid the duplication.

Cache system contains two different nodes one that stores recently submitted complaint files and another node that stores recently accessed complaint files. Storing of files in their respective nodes are performed by cache master. When new file arrives for storing in cache, the origin of the file is identified by the cache master and fed into the respective cache nodes. Cache master must aware of the files that are stored in the cache nodes [10]. It should keep track on files that are stored and deleted from the cache and maintain index file.

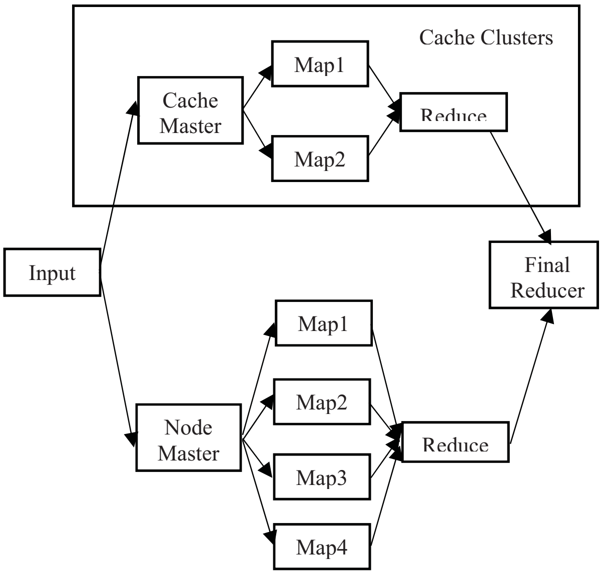

Fig. 2 Illustration on the MapReduce Programming Model with Cache Cluster

C. Map Cache

There are several problems that are caused during designing of the Hadoop MapReduce framework. The first is, when mappers request cache for input? As described cache items are managed by the cache master, when the complaint file is requested for processing initially request will be sent to cache master and data master [1]. The cache master invokes the cache mapper to split the complaint files from the cache and fed those records to the worker nodes in the cache clusters. The cache mapper phase and data mapper phase performs the same operation on different set of complaints and generates the different intermediate result.

D.Reduce Cache

The cache reduce process is more complicated. The first step is to sort and shuffle the intermediate result generated by the map phase and fed into the workers in the reduce phase. Intermediate result from cache map phase and the data map phase are reduced separately. The results from both reducers can be used immediately for processing [5]. Cache map reducers generate the result earlier than data reducers. The cache results are immediately available to the user for processing the complaints. But the cache result is generated only by processing the recently filed complaints and recently accessed complaints. So, cache result provides only partial output. The final result is obtained by re-reducing the result from the cache reduce phase and the data reduce phase. We present an additional reducer that will generate the final output by sorting and shuffling the intermediate result from both reducers.

E. Noise Addition for Security

After the complaints are submitted cache master adds noise value to the actual input file and store it in the Hadoop Distributed File System (HDFS). Therefore, malicious user that aim to leak the information about an individual input or signal its presence in the input dataset will get the strange value. The authorized MapReduce code can able to remove the noise from the input files and process them to generate the actual output [11].

F. Processing Complaints

In the processing of complaints, the system provides complaint automation system. It has three steps namely,

- Internal review
- Actual processing
- External review.

Internal review involves analyzing the complaint and forwarding to the concerned authority. Actual processing refers to the processing and verifying the complaint received. External review involves implementation of the complaint statement and rectifying it.

IV. Performance Evaluation

A. Implementation

We extend Hadoop to implement Cache. Hadoop contains a collection of libraries and tool for Distributed File System (DFS) and MapReduce computation. The complexity of entire package is beyond our control, so we implement the cache technique for Hadoop in a non-intrusive approach and by changing the components that are open to application developers. Basically, the cache manager is implemented as separate server. The cache manager uses HDFS and DFS component of Hadoop to manage storage of cache items.

In order to access cache items, the mapper and reducer first request the cache manager. However this cannot be implemented in Mapper and Reducer classes. Hadoop mapper and reducer classes cannot identify the file split they are working on. Therefore, cache requests cannot be sent from mapper or reducers. We alter InputFormat class which is responsible for splitting the input files to request the cache files. We also alter the TastTracker, which is responsible for managing jobs. TaskTracker is able to understand file split and bypass the execution on mapper classes.

B. Results

Fig. 3 presents the speedup and completion time of the processing. The size of the input data increases and is represented as a percentage number. This complaint processing system is more CPU bound compared to other applications, as a result Cache enabled Hadoop can bypass computation tasks that take more time, which achieves larger speedups. The speedup decreases due to the increasing size of input files, but cache enabled Hadoop can able to complete the job faster than Hadoop in every situation.

The increased number of mapper and reducer improve the processing speed. The cached complaint files and general complaint files are fed into separate MapReduce phase for processing, therefore the result generated faster than the Hadoop. The result from the cache reducers is available immediately before the data reducers completes its computation.

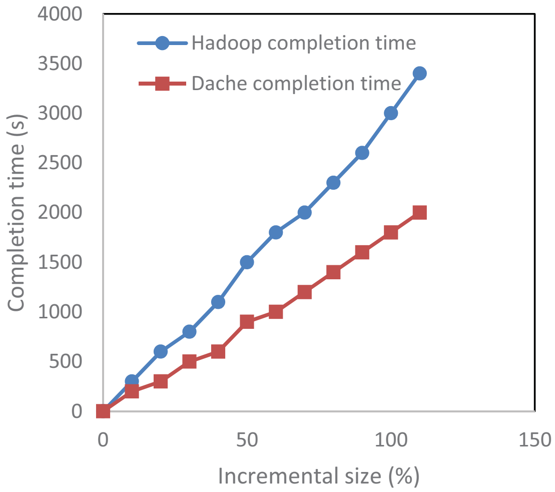

Fig. 3 The speed of Cache over Hadoop and their completion time

V.Conclusion

We present the system includes storing of data in Hadoop platform. Unlike the existing system, the system uses caching technique to provide the efficient handing of complaints. The caching system for big data can bypass computation tasks that take more time, which achieves larger speedups. The system with caching technology generates the reports faster than traditional database systems. It allows accessing the recently filed complaints more quickly than older complaints.

References

[1] Arthur G. Erdman, Daniel F. Keefe, Senior Member, IEEE, and Randall Schiestl, “Grand Challenge Applying Regulatory Science and Big Data to Improve Medical Device Innovation,” IEEE Transaction on Biomedical Engineering, vol. 60(3), pp. 700-706, March 2013.

[2] Benedikt Elser and Alberto Montresor, “An Evaluation Study of BigData Frameworks for Graph Processing,” IEEE International Conference on Big Data, pp.60-67, 2013.

[3] C.Dobre, F.Xhafa, “Intelligent services for Big Data science,” Future Generation Computer Systems, pp. 1-15, July 2013.

[4] C.L. Philip Chen and Chun-Yang Zhang, “Data-intensive applications, challenges, techniques and technologies A survey on Big Data,” Information Sciences, pp. 1-34, January 2014.

[5] Chad A. Steed, Danial M. Ricciuto, and Galen Shipman, “Big data visual analytics for exploratory earth system simulation analysis,” Computers & Geosciences, pp. 71-82, August 2013.

[6] Chia-Wei Lee, kuang-Yu Hsieh, Sun-Yuan Hsieh, and Hung-Chang Hsiao, “A Dynamic Data Placement Strategy for Hadoop in Heterogeneous Environments,” Big Data Research, pp. 14-22, July 2014.

[7] Daniel E. O’Leary, “Artificial Intelligence and Big Data,” IEEE Intelligent Systems, pp. 96-99, March/April 2013.

[8] Foto N. Afrati and Jeffrey D. Ullman, “Optimizing Multiway joins in Map reduce Environment,” IEEE Transaction on Knowledge and Data Engineering, vol. 23(9), pp. 1282-1298, September 2011.

[9] Hadoop, http://hadoop.apache.org/, 2014.

[10] Juwei Shi, Wei Xue, Wenjie Wang, and yuzhou Zhang, “Scalable community detection in massive social networks using MapReduce,” IBM Research and Development, vol. 57(3/4), pp. 1-14, May/July 2013.

[11] Quang Tran and Hiroyuki Sato, “A Solution for Privacy Protection In MapReduce,” IEEE 36th International Conference on Computer Software and Applications, pp. 515-520, 2012.

[12] Yaxiong Zhao, Jie Wu, and Cong Liu, “Dache: A Data Aware Caching for Big-Data Applications Using the MapReduce Framework,” Tsinghuascience and technology, vol. 19(1), pp. 39-50, February 2014.