IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 3, Issue 4, Aug-Sept, 2015
ISSN: 2320 – 8791 (Impact Factor: 2.317)
www.ijreat.org

EHDFS: Overview Of Novel Architecture And Data Retrieval Model In Big Data Scenario

Anitha. R[1], John Bright. A[2], Nikhil Selvaraj[3]

[1]Associate Professor, Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, Pennalur-602117

[2,3]Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, Pennalur-602117

***Abstract* - Recent technological advancements have led to the growth in the volume of data towards zetabyte size over the past decade. This huge volume of data along with huge variety and velocity is termed as big data. Due to its sheer volume of data, efficient retrieval of data with reduced latency plays a major role. This growth of data calls for a novel file system architecture for data retrieval. This paper describes a novel framework for efficient retrieval of data using an Enhanced Hadoop Distributed file System architecture (EHDFS). Performance of queries due to the availability of files for query processing is greatly improved by the efficient use of namenode in the proposed EHDFS and its analysis thereof. Hence this paper proposes a generic approach of using secondary namenode in EHDFS. The novel EHDFS framework addresses the big data challenges. The proposed approach has exploited various methodologies in reducing the latency during data retrieval. This paper investigates the issues on creation of secondary namenode and its management for fast retrieval of data using EHDFS. The proposed architecture is implemented and the experimental results show that our approach works efficient by increasing the throughput and also by handling large number of queries efficiently and thereby reducing the latency. The efficacy of the approach is tested through experimental studies using KDD Cup 2003 dataset. In the experimental results, proposed 'EHDFS' with new methodologies outperforms compared to the existing methods.**

Keywords: *Bigdata, EHDFS, namenode, secondary namenode, bloom filter.*

I. INTRODUCTION

Big data is defined as large amount of data which requires new technologies and architectures so that it becomes possible to extract value from it by capturing and analysis process. Due to such large size of data it becomes very difficult to perform effective analysis using the existing traditional techniques. Big data due to its various properties like volume, velocity, variety, variability, value and complexity put forward many challenges. Since Bigdata is a recent upcoming technology in the market which can bring huge benefits to the business organizations, it becomes necessary that various challenges and issues associated in bringing and adapting to this technology are brought into light. The various challenges and issues in adapting and accepting Bigdata technology, its tools (Hadoop) are also discussed in detail along with the problems Hadoop is facing. The paper concludes good Bigdata practices to be followed in order to process this huge data commonly known as Big Data. The existing technology makes use of a file system called the Hadoop Distributed File System (HDFS) which makes data retrieval easier and faster. Even though the Hadoop Distributed File System comprises of large number of servers, the two main types of servers in HDFS are single NameNode and DataNodes. The NameNode contain links to the DataNodes and also stores information about them. The DataNodes on the other hand are responsible for storing user data and also for performing read, write or update of a file upon instruction from the NameNode. Now failure of the DataNode is more common but this can be dealt with easily since there is always a backup in the form of another DataNode. However, since there is only a single NameNode, its failure leads to loss of all the links to the DataNodes. Hence the failure of any one of these servers can compromise on efficient data storage and retrieval. Thus the main objective of this paper is to deal with the failure of the secondary name node problem which leads to erroneous results. Thus this paper proposes a change to the existing HDFS architecture by introducing a secondary NameNode in addition to the original NameNode which ensures that the links to the DataNodes are not lost at any instant. The novel EHDFS also contributes in proposing the new methodologies for the creation of a secondary Namenode and update of the secondary Namenode as well. The update of secondary Namenode ensures that even in case of the original Namenode failure, the updated information is not lost.

This updation is done using a special kind of a probabilistic data structure called Hadoop Namenode Bloom Filter. These proposals when implemented aims to increase the efficiency of data retrieval in a big data scenario.

The rest of the paper is organized as follows: Section 2 summarizes the related work and the problem statement. Section 3 discusses the detailed design of the system model. Section 4 describes the proposed Hadoop namenode bloom filter details and its structure. The performance evaluation based on the prototype implementation is given in Section 5 and Section 6 concludes the paper.

Problem statement

In a big data environment where the data is huge and geographically distributed, efficient retrieval of data is of prime focus. Major roadblock is latency in retrieval. This paper proposes an efficient system for fast data retrieval in a big data scenario that will have the following as major goals: reduced latency, improved throughput. The following is considered as a minor goal: Reduction in network traffic. The primary contribution of this paper is the Enhanced Hadoop Distributed File System architecture which satisfies the above mentioned objectives. In the proposed research, we have studied the impact of primary name node failure and also the impact of stale data in the secondary name node. Thus the proposed model has developed a mechanism to adaptively minimize such an impact so as to optimize systems performance.

II. RELATED WORK

Recently a large volume of work is being pursued in data analytics in big data [1]. Kyong-Ha et al. has discussed the optimization strategies and open issues and challenges raised. in big data. In paper [2][3] the authors has claimed that the data retrieval using metadata is less when compared to without using metadata due to reduction in latency. Author Chang Liu et al. [4] has discussed that storing replicas at different servers and/or locations will make user data easily recoverable from service failures. He has also proposed a multi-replica Merkle hash tree (MR-MHT) a novel authenticated data structure designed for efficient verification of data updates. Author Yu Hua et al. [5] has proposed a scalable and adaptive metadata management in ultra large scale file systems. Author Kaushik Velusamy et al. [6] has proposed an inverted index data structure is fast and returns all the relevant results. In his work he has established Hadoop cluster and by passing Wikipedia files as input data, inverted indexing is done.Yongqiang He et al. [7] has suggested that Data placement structure is important as it affects warehouse performance. Quanqing Xu et al. [8] has discussed about the efficient and scalable metadata management in large scale databases. Author Bin Lan et al. [9] has proposed a variant of the signature file, called Bit-Sliced Bloom-Filtered Signature File for mining frequent patterns. Many researchers have investigated possible ways to improve the performance of clustering based on the popular clustering algorithms like partition clustering, hierarchical clustering and frequent item based clustering. The concept of bloom filter in cloud computing is explained in depth in the paper[10]. The author has discussed about the independent lookup using CBF in cloud era. Here, we have proposed an effective approach of using the primary name node which stores metadata and also the handling the failure of primary namenode in an efficient way.

III. SYSTEM MODEL

The section 3 discusses in detail about the existing HDFS architecture and the proposed EHDFS architecture.

3.1 Existing HDFS Architecture

Figure 1 represents the existing Hadoop Distributed File System architecture.

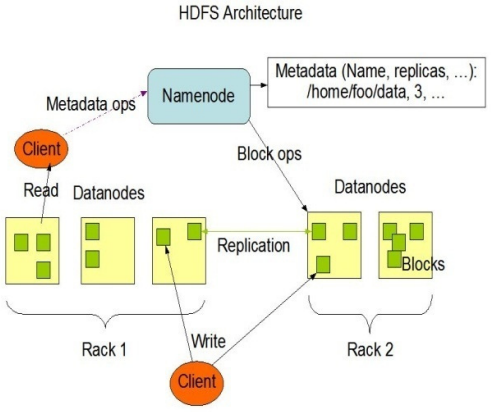

Figure.1 Hadoop Distributed File System Architecture

The existing HDFS has master/slave architecture. The HDFS cluster consists of a single namenode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of datanodes, usually one per node in the cluster, which manages the storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of datanodes. The namenode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to datanodes. The datanodes are responsible for serving read and write requests from the file system's clients. The datanodes also perform block creation, deletion, and replication upon instruction from the namenode. The existence of a single namenode in a cluster greatly simplifies the architecture of the system. The namenode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the namenode. The namenode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the namenode. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. This information is stored by the NameNode. In the existing HDFS when the namenode gets failed then the entire data retrieval process will become tedious. Hence in order to overcome this issue we have proposed EHDFS, an Enhanced Hadoop Distributed File System architecture.

3.2 PROPOSED EHDFS ARCHITECTURE

Figure 2 represents the proposed Enhanced Hadoop Distributed File System Architecture. The proposed EHDFS architecture consists of a secondary namenode, takes care of replication and update of metadata in case of primary namenode failure. The proposed Hadoop Namenode Bloom Filter (HNBF) takes care of the update of the secondary name node. HNBF provides the protocol for updating the metadata information available in the secondary name node in order to maintain the consistency of the metadata file in the secondary name node. In proposed architecture a secondary namenode is introduced to store the replica of original namenode. The failure of primary namenode will automatically direct the user's request to secondary namenode. This novel work reconsiders the design of the existing namenode server and proposes a novel namenode model that reduces the time consumption process by means of implementing a novel data structure called Hadoop Namenode Bloom Filter (HNBF). The model also improves the metadata consistency, in both the primary and secondary namenode by metadata management policies using HNBF. HNBF enhances the performance of the existing HDFS architecture model further. The metadata management is taken care by the proposed HNBF which reduces the number of small writes, and hence significantly improves the overall system performance.

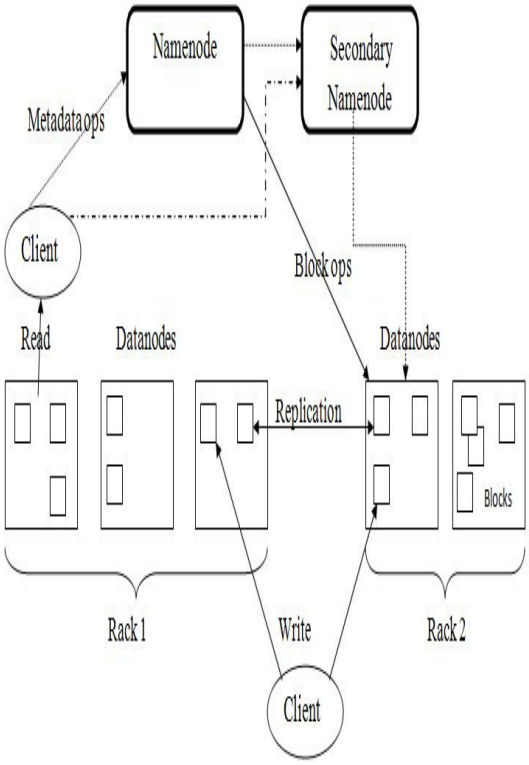

Figure. 2 Enhanced Hadoop Distributed File System Architecture

Namenode reliability results in efficient data finding by providing locality of reference in large scale storage and improves the efficiency of the proposed model. The functionalities of the proposed architecture are described in the below sections.

3.3 SECONDARY NAMENODE MODEL

Figure3 represents the internal architecture of secondary namenode model.

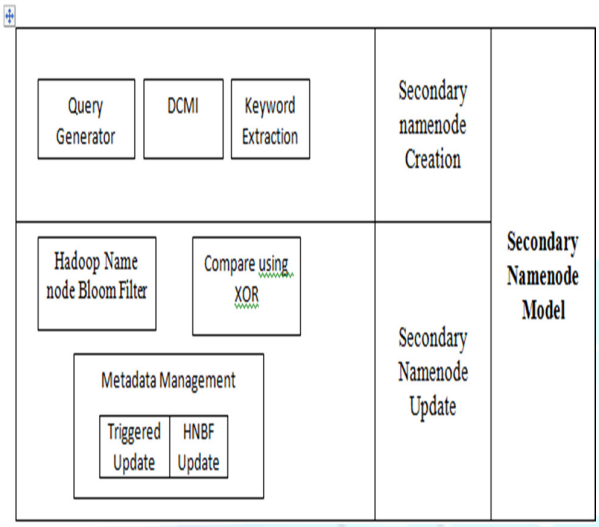

Figure. 3 Internal Architecture of Secondary Namenode Model

It has 2 layers, one for secondary namenode creation and the other for secondary namenode update. In the creation layer the content for the namenode is created. In the update layer, a novel Hadoop Namenode Bloom Filter is proposed and is used to compare the values in the Hadoop namenode Bloom Filter present in both the primary and the secondary namenode using XOR operation and thereby update the secondary namenode. The 2 update policies which are used are triggered update and HNBF update.

3.3.1 CREATION OF SECONDARY NAMENODE

Creation of secondary name node plays a major role in novel architecture. The content of the primary name node is replicated into secondary name node. In case of primary node failure the secondary name node takes care of the request from the user. In such a scenario which is very rare, all the links to the datanodes will be directed to the secondary namenode.

3.3.2 Update of secondary namenode using HNBF

HNBF is a dynamic layered bloom filter. The number of layer represents the number of attribute in the metadata file. Every metadata server has its own Hadoop namenode Bloom filter which is a part of extended bloom filter. The frame of the layer is comprised of suffix, header, counter and body. The suffix information is to group the metadata file of same replica location. The header information is used to identify the attribute.

IV. HADOOP NAMENODE BLOOM FILTER

In this HNBF, each layer has an independent modified bloom filters. The attributes assigned to these layers are independently hashed using MD5, SHA1 and SHA2 algorithms. Initially all the bit values are set to zero. When a metadata file is stored in the metadata server, the respective attributes are hashed and the information's are stored into the respective layers of Hadoop namenode bloom filter. The schemas are identified by their respective header values. Any changes in the primary node are updated to the secondary namenode by using the proposed update policies.

4.1 Update of Secondary namenode using HNBF

An efficient update mechanism deals with two issues. The first issue is when and what kind of update is required and second issue is the amount of bandwidth consumed in maintaining consistency. In order to make the proposed system work perfect with consistent data, the update has to be carried out in two different sequences. 1. Whatever changes occurred to the original file has to be updated to the metadata file and 2. The change in the metadata has to be updated to its replica location. In the proposed model the update is triggered based on threshold value. Updating the metadata file continuously at runtime is expensive. In the Hadoop namenode bloom filter the schemas are represented in different layers. Whenever there exists any change in the layers of Hadoop namenode bloom filter, then the corresponding layer has to be spread to all the metadata servers which hold the replica of that particular file. Hence in spite of updating the whole of the metadata file only the corresponding bit level is updated which reduces the network traffic and also reduces bandwidth consumption.

4.2 Update Mechanism.

The update mechanism of the attributes is carried out efficiently by comparing the value of primary HNBF with secondary HNBF. When changes

occur in the metadata then the corresponding LBF gets changed. The HNBF value of the schema exists in primary location is XOR with the secondary location. When the output converges to one, means that there is a difference in the original and replicated metadata. Hence update is taken place.

Hadoop Namenode Bloom Filter: Algorithm for Update Mechanism
Input: Upload changes in metadata file.
Output: Updated metadata file in Replica Location
/* updating the Hadoop namenode Bloom Filter */
Begin
Update metadata (S1) /* S1 refers to a Single attribute in metadata */

a = old value (HNBF in primary namenode)
b = old value (HNBF in secondary namenode)
X = a (XOR) b;
If (X =1) then
Update (HNBF in replica location);
End

Algorithm for Update Mechanism :
This kind of update reduces the network traffic and time by updating only the respective layer instead of all the attributes. The total storage requirement of HNBF is negligible. Hence this ideal solution to the metadata update problem would provide immediate stability and consistency of all metadata updates with less performance overhead, and no special hardware support.

V. IMPLEMENTATION AND RESULTS

The experiments have been carried out in a Hadoop setup which contains primary name node and the secondary namenode. In the experiment, files are uploaded into the storage and then downloaded based on the user's requirement with and without metadata. In experiment #1 we have used the KDD Cup dataset and investigated the effect of file access performance using metadata with respect to the response time. Figure 4 compares the response time for accessing the file with metadata and without using metadata.

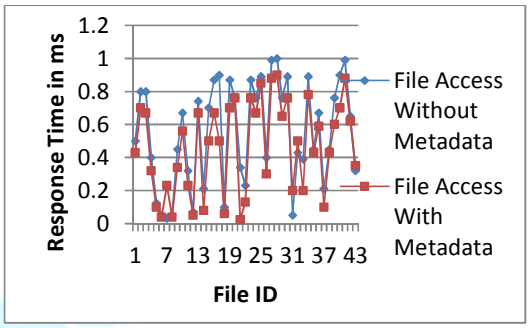

Fig.ure 4. Comparison with respect to response time of file retrieval with and without using metadata

The performance result shows that the response time for retrieving a file using metadata is less when compared to that without using metadata because the metadata attribute holds the information about where the data is stored. Hence when the metadata file is found, the query is mapped to the exact location of the data in the data server, leads to the speedy retrieval of data from the data servers.

In experiment #2 we have used the same dataset and the experiment is conducted for updating the metadata in the replica location using Hadoop namenode bloom filter. Figure 5 compares the update time of metadata in replica location with and without using Hadoop namenode bloom filter. Response time is the time taken to update the metadata in the replica location using LBF. Using LBF the update is carried out only for the respective attribute which is modified, hence the time taken is less.

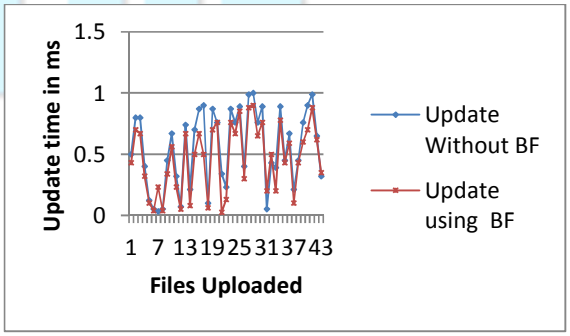

Figure 5. Comparison of update time with and without using BF

Experiment#3 is carried out to show the performance of HNBF by means of percentage of error rate which is calculated using the amount of stale data provided to the user instead of the updated data. We observe that periodic update has higher error rates than HNBF update as shown in Figure 6. This could be explained as follows. Assume that when an user modifies the file at time t1 which gets reflected in the metadata attribute, the same has to be updated in the replica location otherwise the stale metadata misleads the communication towards data server. In this experiment the periodic update is calculated for every 20 minutes.

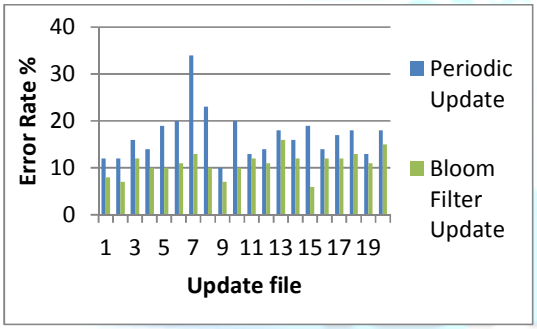

Figure 6. Comparisons of Error Rate% of Bloom Filter update with periodic update

Experiment#4 is carried out to show the performance of bloom filter by means of False alarm which is calculated using the number of times the update takes place for the same existing data. From the figure7 shown below it is observed that bloom filter update results in the lesser number of false alarms compared with the triggered update and periodic update. The reduction of false alarms by bloom filter based update is mainly due to their threshold fixing capability.

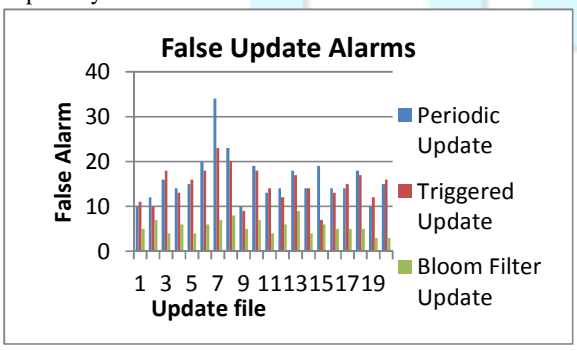

Figure 7. Comparisons of False Alarm update time with and without using HNBF

The threshold value for BF update depends on the update probability. When there is an increase in the update probability the false alarm is reduced but there will be an increase in the error rate percentage. In periodic update the refresh time is fixed for every 20 minutes but the probability of update has an effect of changing the refresh time of the attribute thus, decreasing the false alarms. Hence the above experimental results demonstrate that our EHDFS design is highly effective and efficient in improving the performance of data retrieval in bigdata environment.

VI. CONCLUSION

This paper presents the data retrieval framework in bigdata environment that can be used to store and retrieve data in a bigdata environment using Hadoop distributed file system. Due to huge amount of data stored in bigdata scenario and due to lack of information about the data (i.e. Location, file name, size of the file ... etc.) the data cannot be retrieved much efficiently. In order to make the system work efficiently, a novel architecture with a new design called enhanced hadoop distributed file system architecture is proposed. Further, more specifically, the proposed EHDFS architecture comprises of two levels of name node. The first level can be viewed as a filter that works at the speed of traffic by quickly providing a replica which can potentially match the input and the second level take care of the update of that actual match. Finally, with these features the proposed work tends to give an Enhanced Hadoop Distributed File System Architecture**.** What does the future hold? Many further researches are ongoing in some aspects related to data retrieval in big data. Our current implementation is weak in security considerations in HDFS. Finally, with all these feature the future work tends to give a SHDFS - Secured HDFS where data lake process can be included.

REFERENCES

[1] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi Yon Dohn Chung, Bongki Moon, “Parallel Data Processing with MapReduce: A Survey”, SIGMOD Record, Vol. 40, No. 4, 2011.

[2] R. Anitha, SaswatiMukherjee. “A Dynamic Semantic Metadata Model in Cloud Computing”, Proc. of Springer CCIS, Vol.2, pp.13–21, 2012.

[3] Hua, Y. Jiang, H. Zhu, Y. Feng, D. Tian, L."Semantic-aware Metadata Organization Paradigm In Next-generation file Systems", IEEE Transactions On Parallel Distributed Systems, Vol. 23, No. 2, pp.337–344, 2012.

[4] Chang Liu, Rajiv Ranjan, Chi Yang, Xuyun Zhang, Lizhe Wang, Jinjun Chen, "MuR-DPA: Top-down Levelled Multi-replica Merkle Hash Tree Based Secure Public Auditing for Dynamic Big Data Storage on Cloud", International Association for Cryptologic Research, pp.391-396, 2014.

[5] Yu Hua, Yifeng, Hong Jiang, Dan Feng, and Lei Tian,"Supporting Scalable and Metadata Management in Ultra Large Scale File Systems", IEEE Transactions on Parellel and Distributed Systems, Vol.22, No.4, 2011.

[6] Kaushik Velusamy, Deepthi Venkitaramanan, Nivetha Vijayaraju, Greeshma Suresh, Divya Madhu, "Inverted Indexing In Big Data Using Hadoop Multiple Node Cluster", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 4, No. 11, 2013 .

[8] Quanqing Xu, Rajesh Vellore Arumugam, Khai Leong Yong, and Sridhar Mahadevan, "Efficient and scalable Metadata Management in EB-scale File Systems", IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No.1, pp.1-10, 2013.

[9] Bin lan. Beng chin ooi, kian-lee tan, "Efficient Indexing structure for mining frequent patterns", in proc. International conference on Data Engineering, 2002, pp. 453-462.

[10] R. Anitha, SaswatiMukherjee. "CBF: Metadata Management in Cloud Computing", Proc. of International Conference on Computational Intelligence and Information Technology, 2013.