ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

Cloud Based Adaptive Overlapped Data Chained Declustering

Vidya G. Shitole[1], Prof. N. P. Karlekar[2]

Student, M.E. 2nd year, Computer Engineering, SIT Lonavala, University of Pune, Maharashtra, India[1]

Associate Professor, Computer Engineering, SIT Lonavala, University of Pune, Maharashtra, India[2]

Abstract: Distributed file systems (DFS) are key building blocks for cloud computing applications based on the Map Reduce programming paradigm. In such file systems, nodes simultaneously serve computing and storage functions; a file is partitioned into a number of chunks allocated in distinct nodes so that Map Reduce tasks can be performed in parallel over the nodes. However, in a cloud computing environment, failure is the norm, and nodes may be upgraded, replaced, and added in the system. Files can also be dynamically created, deleted, and appended. This results in load imbalance; that is, the file chunks are not distributed as uniformly as possible in the nodes. Although distributed load balancing algorithms exist in the literature to deal with the load imbalance problem, emerging DFS in production systems strongly depend on a central node for chunk reallocation. The performance of the proposal implemented in the Hadoop distributed file system is further investigated in a cluster environment.

Keywords: Load balance, Distributed file systems, Clouds, Map Reduce

I. INTRODUCTION

Cloud computing is a compelling technology. In clouds, clients can dynamically allocate their resources on-demand without sophisticated deployment and management of resources. Key enabling technologies for clouds include the Map Reduce programming paradigm, distributed file systems, virtualization, and so forth. These techniques emphasize scalability, so clouds can be large in scale, and comprising entities can arbitrarily fail and join while maintaining system reliability. Load balance among storage nodes is a critical function in clouds. Here, the load of a node is typically proportional to the number of file chunks the node possesses.

The resources in a load-balanced cloud can be well utilized and provisioned, maximizing the performance of Map Reduce based applications. State-of-the-art distributed file systems (e.g., Google GFS and Hadoop HDFS) in clouds rely on central nodes to manage the metadata information of the file systems and to balance the loads of storage nodes based on that metadata. The centralized approach simplifies the design and implementation of a distributed file system. (e.g., Google GFS and Hadoop HDFS) in clouds rely on central nodes to manage the metadata information of the file systems and to balance the loads of storage nodes based on that metadata. The centralized approach simplifies the design and implementation of a distributed file system.

However, recent experience e.g., concludes that when the number of storage nodes, the number of files and the number of accesses to files increase linearly, the central nodes (e.g., the master in Google GFS) become a performance bottleneck, as they are unable to accommodate a large number of file accesses due to clients and Map Reduce applications. Consequently, tackling the load imbalance problem with the central nodes only serves to increase their heavy loads, especially considering the load rebalance problem is NP-hard. Moreover, the central nodes may be the single point of failure; if they fail, then the entire file system crashes.

The proposal is assessed through mathematical analysis, computer simulations and a real implementation in Hadoop HDFS. The performance results indicate that although each node performs the load rebalancing algorithm independently without acquiring global knowledge, the proposal is comparable with the centralized approach in Hadoop HDFS and remarkably outperforms the competing distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead.

II. OBJECTIVE

The objective in the current study is to design a load rebalancing algorithm to reallocate file chunks such that the chunks can be distributed to the system as uniformly as possible while reducing the movement cost, which is defined as the number of chunks, migrated to balance the loads of the chunk servers, as much as possible. Specifically, our load rebalancing algorithm aims to minimize the load imbalance factor in each chunk server.

A. Primary objective is to store data reliably.
B. MapReduce is used for processing of data & faster retrieval of stored data.

III. HADOOP MAPREDUCE

Hadoop is the most popular tool for content classification in the www search and a software platform where it is easier to develop and deal with large-scale data. Hadoop has a reliable, efficient and scalable way to process data. Being reliable means that Hadoop is able to maintain multiple copies of data and can automatically re-deploy computing tasks after the failure of the maindata. Being efficient means that Hadoop works through parallel way to speed up processing. In addition, the usage cost of Hadoop

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

is low because it can use a general machine server farm to distribute and process data. The server farms can reach hundreds of nodes, which anyone can use. Therefore it has a large scalability.

Hadoop is an open source distributed parallel computing platform. It is mainly composed of two parts: the MapReduce algorithm implementation and a distributed file system. MapReduce algorithm comes from functional programming ,and it is natural to construct the algorithm .

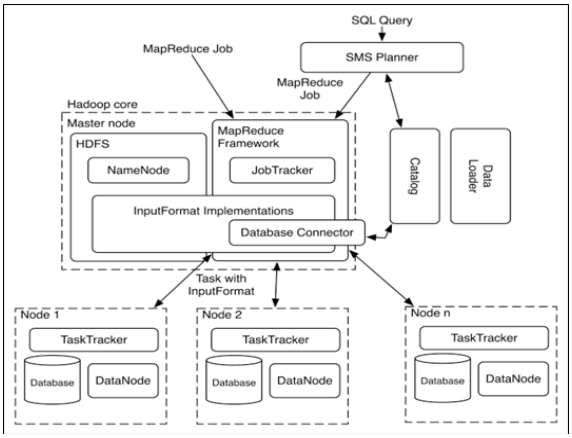


Fig. 1 The Architecture of Hadoop DB

Hadoop is mainly composed by three parts of (the Hadoop Distributed the File System) HDFS, MapReduce, and Hbase. On the bottom is HDFS, it stores the files on all storage nodes in the Hadoop cluster. Above the HDFS layer is the MapReduce engine, which consists of operating server and task server; HBase is another interesting applications on the top of HDFS.

HBase differs a lot from the traditional relational database. It is a distributed database based on column storage model, which is similar to Google's BigTable high-performance database system.

HDFS architecture is based on a specific set of nodes, the metadata nodes and the data nodes, shown in Figure1.HDFS includes a single NameNod. The Namenode is used to manage the file system namespace and it saves the metadata of all files and folders in a file system tree; that Namenode provides internal metadata service in HDFS; DataNode provides HDFS with storage blocks, and DataNode is the real place to store data in the file system. The client or the namenode can request data nodes to write or read data block. The datanode periodically returns its stored data block information to metadata node.

A. MapReduce Framework

MapReduce framework is responsible for automatically splitting the input, distributing each chunk to workers (mappers) on multiple machines, grouping and sorting all intermediate values associated with the intermediate key, passing these values to workers (reducers) on multiple resources, this is shown in Fig.2. Monitoring the execution of mappers and reducers as to re-execute them when failures are detected is done by the master.

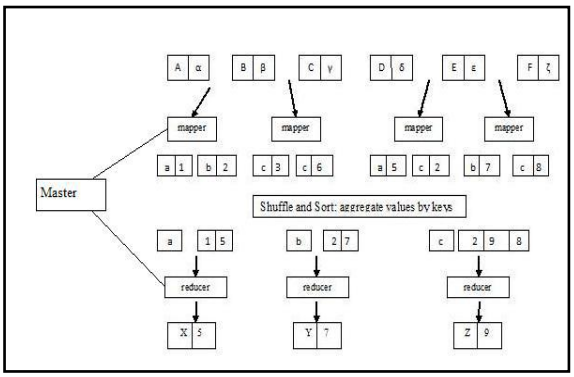


Fig. 2 Simplified view of MapReduce

B. Implementation of VDB With Hadoop MapReduce

In order to improve the performance efficiency of the VDB the Hadoop MapReduce is added at the executor phase. The executor will pass the mapper's sub query to the Master of the MapReduce. The master will automatically split the input into chunks (splits) and finds M mappers and R reducers. The splits can be processed in parallel by the mappers. Reduce invocations are distributed by partitioning the intermediate key space into R pieces using a partitioning function. The number of partitions (R) and partitioning functions are specified by the user.

The output of the R Reducers stored in R output files. This output files will fit our needs. This is shown in Fig.3. The output will be sent back to the user.

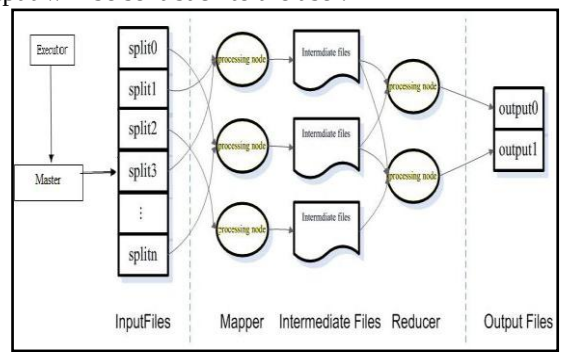


Fig. 3 MapReduce execution flow with VDB

C. Hadoop MapReduce Algorithm

MapReduce program, the first step is called For the map, the Map function works as follows: through the Mapping function some data elements become input data one at a time, and the Mapping will spread each mapping result separately to an output data elements. Mapping creates a new list of output data by applying a function to each element in the list of input data.

The second step of the MapReduce program is called Reduce (aggregation); implementation process of the reduce function: Reducing (aggregate) function allows the data to aggregate together. Reducer function receives iterator from the list of the input, aggregates these data, and then returns an output value.

The third step of the MapReduce program is to put map and Reduce together on the MapReduce, with a key

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

associated with each value. For values related to the key, no value is separate in MapReduce. mapping and reducing function not only receive values (Values), but (key, value) pairs. The output of each of these functions are the same, being a key and a value, which will be sent to the next list of the data stream.

IV. THE LOAD REBALANCING PROBLEM

To simplify load rebalancing first assume a homogeneous environment, where migrating a file chunk between any two nodes takes a unit movement cost and each chunkserver has the identical storage capacity.

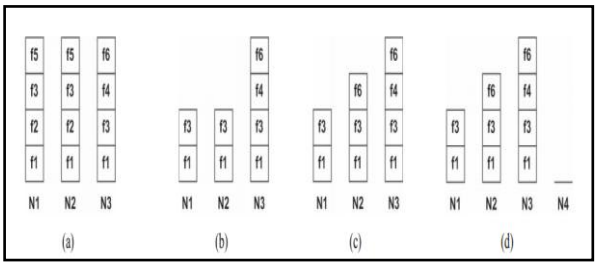


Fig. 4 An example illustrates the load rebalancing problem

Where,

a) An initial distribution of chunks of sixfiles f1, f2, f3, f4, f5 and f6 in three nodes N1, N2 and N3,
b) Files f2 and f5 are deleted,
c) f6 is appended, and
d) Node N4 joins. The nodes in (b), (c) and (d) are in a load-imbalanced state.

A. Algorithm Overview

A distributed file system is in a load-balanced state if each chunkserver hosts A chunks. Thus, for a large-scale distributed file system, each chunkserver nod ei first estimates without global knowledge in our proposed algorithm whether it is light or heavy.

By a light (orunderloaded) node, we mean that the number of chunks the node hosts is smaller than the threshold of the average $(1-\Delta L)A$, where ΔL is a system parameter and $0 \le \Delta L < 1$. In contrast, a heavy (or overloaded) node manages the number of chunks greater than $(1+\Delta U)$ A, where $0 \le \Delta U < 1$.

Conceptually, this algorithm proceeds as follows. Consider any node $i \in V$. If node i is light, then it seeks a heavy node and takes over at mostAchunks from the heavy node. Specifically, ifi is the least-loaded node in the system, i has to leave the system by migrating its locally hosted chunks to its successor i+1and then rejoin instantly as the successor of the heaviest node (say, node j), that is, j+1.

B. The Properties Of Reallocation

a. Low Movement Cost

As i is the lightest node among all chunkservers, the number of chunks migrated because of i's departure is minimal, introducing the minimum movement cost.

b. Fast Convergence Rate

Node i seek to relieve the load of the heaviest node j, hoping that the system converges quickly towards a load-balanced state. i allocates A chunks from the heaviest node j if j's load exceeds 2A;otherwise,i requests the load of $Lj-A$ from j. i then becomes load-balanced if $Li = A$, and the load of j is immediately relieved

Possibly, j remains the heaviest node in the system even if it has migrated its load to i. If so, among all light nodes, the least-loaded one departs and then re-joins the system as j's successor. That is, i become node j+1, and j's original neighbour i thus becomes node j+2.Such a process repeats iteratively until j is no longer the heaviest. Subsequently, among the remaining heavy nodes, the heaviest one relieves its load by having the lightest node re-join as its successor.

V. THE LOAD REBALANCING ALGORITHMS

A. Algorithm 1: SEEK (V,ΔL,ΔU): a light node i seeks an overloaded node j

Input: vector V={samples}, ΔL and ΔU
Output: an overloaded node, j

Step1: Ai an estimate forAbased on { Aj :j∈ V}
Step 2: if Li <(1−ΔL) Ai then
Step 3: V←V ∪ {i};
Step 4: sort V according to Lj(∀ j∈ V)in ascending order;
Step 5: k←i's position in the ordered set V;
Step 6: find a smallest subset P ⊂ V such that
(i) Lj >(1 + ΔU) Aj,∀ j∈ P,and
(ii) j ∈ P
Step 7: j←the least loaded node in P;
Step 8: return j;

B. Algorithm 2: MIGRATE(i, j): a light node i requests chunks from an overloaded nodej

Input: a light node i and an overloaded node j

Step1: if Lj > (1 + ΔU) Aj and j is willing to share its load with i then
Step2: I migrates its locally hosted chunks to i+1;
Step3: i leaves the system;
Step4: i re-join the system as j's successor by having i←j+1;
Step5: t← Ai;
Step6: if t > Lj−(1 + ΔU) Ai then
Step7: t←Lj−(1 + ΔU) Ai;
Step8: i allocates t chunks with consecutive IDs from j;
Step9: j removes the chunks allocated to i and renames its ID in response to the remaining chunks it manages.

Algorithm 1: specifies the operation that a light node i seeks an overloaded nodej, and
Algorithm 2: shows that i requests some file chunks from j

C. Goals of Load Balancing

a. To improve the performance.
b. To maintain the system stability.
c. To increase the flexibility of system.
d. To have a backup plan in case system fails even partially

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

D. The Basic Concepts Include In Load Balancing

a. Clustering and Declustering
b. Chained-based Declustering.
c. Balancing Access Loads
d. Partitioning
e. Vertical Partitioning
f. Disk Failures and Space Utilization

E. Load Balancing Policies

1. Location Policy:

The policy used by a processor or machine for sharing the task transferred by an over loaded machine is termed as Location policy.

2. Transfer Policy:

The policy used for selecting a task or process from a local machine for transfer to a remote machine is termed as Transfer policy.

3. Selection Policy:

The policy used for identifying the processors or machines that take part in load balancing is termed as Selection Policy.

4. Information Policy:

The policy that is accountable for gathering all the information on which the decision of load balancing is based ID referred as Information policy.

5. Load estimation Policy:

The policy which is used for deciding the method for approximating the total work load of a processor or machine is termed as Load estimation policy.

6. Process Transfer Policy:

The policy which is used for deciding the execution of a task that is it is to be done locally or remotely is termed as Process Transfer policy.

7. Priority Assignment Policy:

The policy that is used to assign priority for execution of both local and remote processes and tasks is termed as Priority Assignment Policy.

8. Migration Limiting Policy:

The policy that is used to set a limit on the maximum number of times a task can migrate from one machine to another machine.

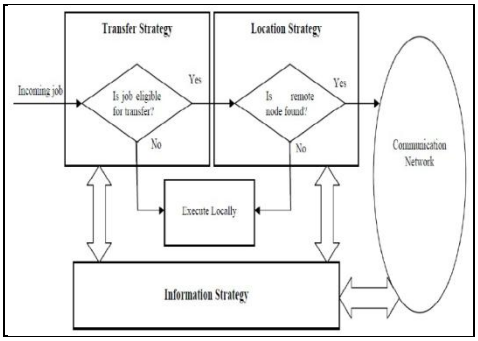


Fig. 5 Interaction among components of a load balancing algorithm

Dynamic load balancing algorithms, the current state of the system is used to make any decision for load balancing. It allows

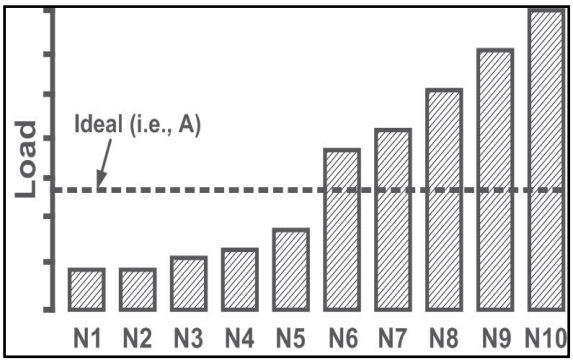
For processes to move from an over utilized machine to an under-utilized machine dynamically for faster execution as shown in Figure 5. This means that it allows for process preemption which is not supported in Static load balancing approach. An important advantage of this approach is that its decision for balancing the load is based on the current state of the system which helps in improving the overall performance of the system by migrating the load dynamically

F. Algorithm Implementation

The system performs Algorithms 1 and 2 simultaneously without synchronization. It is possible that a number of distinct nodes intend to share the load of node j (Line 1 of Algorithm 2. Thus, j offloads parts of its load to a randomly selected node among the requesters. Similarly, a number of heavy nodes may select an identical light node to share their loads. If so, the light node randomly picks one of the heavy nodes in the reallocation.

Without global knowledge, pairing the top - k1 light nodes with the top - k2 heavy nodes is clearly challenging. In-stead, we let each light node I estimate its k value based on its sample set (Line 5 in Algorithm 1).i re-joins as a successor of a heavy node j, where -j is the least loaded among the (estimated) top-k2 heavy nodes;

-The total exceeding load of these top-k2 heavy nodes is approximately greater than k times Ai (Line 6 in Algorithm 1).



(a)

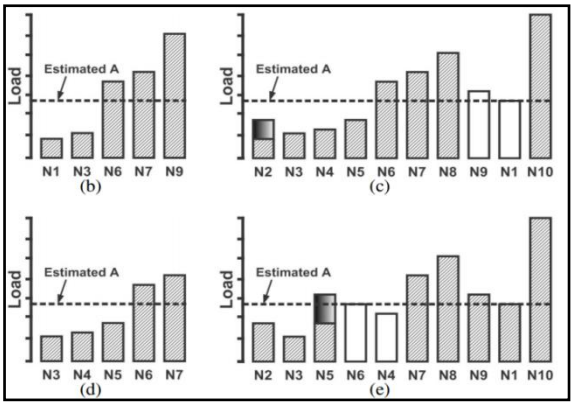


Fig. 6 An example illustrating algorithm

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

Where,

a. The initial loads of chunkservers N1, N2,· · ·,N10,
b. N1samples the loads of N1, N3, N6, N7, and N9 in order to perform the load rebalancing algorithm,
c. N1leaves and sheds its loads to its successor N2, and then re-joins as N9's successor by allocating A N1chunks (the ideal number of chunks N1estimates to manage) from N9,
d. N4 collects its sample set {N3,N4,N5,N6,N7},and
e. N4departs and shifts its load to N5, and it then re-joins as the successor of N6 by allocating N6 AN4 chunks from N6

Each node in our proposal implements the gossip-based Each node in this proposal implements the gossip-based aggregation protocol to collect its set V and estimate A (Line 1 in Algorithm 1). Precisely, with the gossip-based protocol, the participating nodes exchange locally maintained vectors, where a vector consists of s entries, and each entry contains node ID and node network address representing a node selected randomly in the system.
The nodes perform load rebalancing algorithm periodically, and they balance their loads and minimize the movement cost in a best-effort fashion.

VI. MANAGING REPLICAS

In distributed file systems (e.g. Google GFS and Hadoop HDFS), a constant number of replicas for each file chunk are maintained in distinct nodes regardless of node failure and departure to improve file availability. The current load balancing algorithm does not treat replicas distinctly. It is unlikely that two or more replicas are placed in an identical node because of the random nature of our load rebalancing algorithm. More specifically, in this proposal each under loaded storage node samples a number of nodes, each selected with a probability of 1/ n, to share their loads (where n is the total number of storage nodes).
Given k replicas for each file chunk (where k is typically a small constant, and k = 3 in GFS), the probability that k_ replicas ($k \leq k$) are placed in an identical node due to migration of our load balancing algorithm is (1/n) ^k' independent of their initial locations. For example, in a large-scale distributed file system with n = 1, 000 storage nodes and k = 3, then the probabilities are only(1/10)^6 and (1/10)^9 for two and three replicas, respectively, installed in the same node. Consequently, the probability of more than one replica appearing in a node.

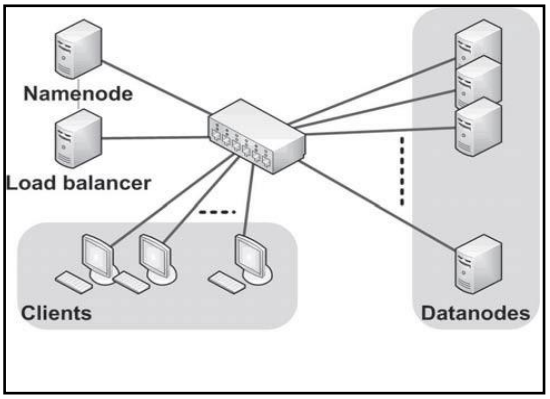


Fig. 7 The setup of the experimental environment

The Load Balancing Algorithm strives to balance the loads of nodes and reduce the demanded movement cost as much as possible, while taking advantage of physical network locality and node heterogeneity. In the absence of representative real workloads (i.e., the distributions of file chunks in a large-scale storage system) in the public domain, we have investigated the performance of system and compared it against competing algorithms through synthesized probabilistic distributions of file chunks. The synthesis workloads stress test the load balancing algorithms by creating a few storage nodes that are heavily loaded.

VII. METRICS FOR LOAD BALANCING

The different qualitative metrics or parameters that are considered important for load balancing in cloud computing [8] are discussed as follows:

A. ***Throughput:*** The total number of tasks that have completed execution is called throughput. A high throughput is required for better performance of the system.

B. ***Associated Overhead:*** The amount of overhead that is produced by the execution of the load balancing algorithm. Minimum overhead is expected for successful implementation of the algorithm.

C. ***Fault tolerant:*** It is the ability of the algorithm to perform correctly and uniformly even in conditions of failure at any arbitrary node in the system.

D. ***Migration time:*** The time taken in migration or transfer of a task from one machine to any other machine in the system. This time should be minimum for improving the performance of the system.

E. ***Response time:*** It is the minimum time that a distributed system executing a specific load balancing algorithm takes to respond.

F. ***Resource Utilization:*** It is the degree to which the resources of the system are utilized. A good load balancing algorithm provides maximum resource utilization.

G. ***Scalability:*** It determines the ability of the system to accomplish load balancing algorithm with a restricted number of processors or machines.

H. ***Performance:*** It represents the effectiveness of the system after performing load balancing. If all the above parameters are satisfied optimally then it will highly improve the performance of the system.

VIII. ANALYTICAL MODEL

The load balancing algorithm is a randomized algorithm, in which each node samples a number of nodes independently and uniformly at random. It is possible for multiple light nodes (denoted by the set I) to contend for allocating file chunks from the same heavy node (say, node j). As a result, j migrates parts of its load to a randomly selected node in I. Similarly, multiple heavy nodes may simultaneously have the same light node share

their loads. If so, the light node randomly selects one of the heavy nodes in the reallocation. Consequently, it is essential to study the number of algorithmic rounds after which all light nodes can request loads from heavy nodes.

IX. ADVANTAGES & LIMITATIONS

A. Advantages

1. It used to handle large amounts of work across a set of machines.
2. Enables applications to work with thousands of nodes.
3. It can be used as an open source implementation.
4. MapReduce is a new framework specifically designed for processing huge datasets on distributed sources.
5. Map and Reduce techniques to break down the parsing and execution stages for parallel and distributed processing.
6. The cluster machines can read the data set in parallel and provide a much higher throughput.

B. Limitations

1. It cannot update data after it is inserted
2. There is no "insert into table values ... " statement
3. It can only load data using bulk load
4. There is not "delete from " command
5. It can only do bulk delete

X. FUTURE SCOPE

A. Video Streaming-Video streaming means mobile nodes view a video stored in the main cloud. If a cloudlet architecture is used, mobile users can view the cached video from their cloudlet.

B. Implementing mapreduce technique in VDS

C. Implement the survey application for Hadoop.

D. To distributes streaming media content, both live and on-demand to users who cooperate in the streaming

XI. CONCLUSION

The goal of load balancing is to increase client satisfaction and maximize resource utilization and substantially increase the performance of the cloud system. The Load Rebalancing Algorithm strives to balance the loads of nodes and reduce the demanded movement cost as much as possible. In the absence of representative real workloads (i.e., the distributions of file chunks in a large-scale storage system) in the public domain, it need to investigate the performance of proposal and compared it against competing algorithms through synthesized probabilistic distributions of file chunks. The synthesis workloads stress test the load balancing algorithms by creating a few storage nodes that are heavily loaded. The performance results with theoretical analysis, computer simulations and a real implementation are encouraging, indicating that our proposed algorithm performs very well. The Load Rebalancing Algorithm is comparable to the centralized algorithm in the Hadoop HDFS production system and dramatically outperforms the competing distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead.

ACKNOWLEDGMENTS

I would like to thank my project guide Prof. N.P. Karlekar, for giving me this chance to explore and increase my knowledge. I would like to thank different journals and blogs of various author and computer scientists, which helped me to gain knowledge about this topic. Last but not the least I thank my beloved parents, friends and well-wishers who helped me by giving various advices regarding the topic by their kind help and assistance

REFERENCES

[1] Hsueh-Yi Chung, Che-Wei hang Hung-Chang Hsiao, Yu-Chang Chao,"The Load Rebalancing Problem in Distributed File Systems,“, 2012

[2] Hung-Chang Hsiao, Hsueh-Yi Chung, Haiying Shen, Yu-Chang Chao,"Load Rebalancing for Distributed File Systems in Clouds,", 2012.

[3] Wenqiu Zeng, Ying Li, Jian Wu, Qingqing Zhong, Qi Zhang, "Load rebalancing in Large-Scale Distributed File System,” 2009.

[4] L.Kiran kumar,V.Ranjith kumar, "Application of Hadoop MapReduce Technique to Virtual Database System Design," 2011.

[5] Fei Hu ,Jim Ziobro, Jason Tillett, Neeraj K. Sharma,"CATCH: A Cloud-based Adaptive Data Transfer Service for HPC," 2011.

[6] Saba Sehrish, Grant Mackey, Pengju Shang, Jun Wang, "Supporting HPC Analytics Applications with Access Patterns Using Data Restructuring and Data-Centric Scheduling Techniques in MapReduce," 2013.

[7] Bin Wu, Shengqi Yang, Haizhou Zhao, and Bai Wang, “A Distributed Algorithm to Enumerate All Maximal Cliques in MapReduce,”

[8] Suresh M., Shafi Ullah Z., Santhosh Kumar B., “An Analysis of Load Balancing in Cloud Computing “2013

[9] R. X. T. and X. F. Z.” A Load Balancing Strategy Based on the Combination of Static and Dynamic, in Database Technology and Applications (DBTA)”, 2010 2nd International Workshop 2010

[10] Abhijit A. Rajguru, S.S. Apte, “A Comparative Performance Analysis of Load Balancing Algorithms In Distributed Systems Using Qualitative Parameters”, International Journal of Recent Technology and Engineering, Vol. 1, Issue 3, August 2012.

[11] Eager, D., Lazowska, E., and J. Zahorjan, ”Adaptive Load Sharing in Homo- geneous Distributed Systems,” IEEE Transactions on Software Engineering, Vol. SE-12, No. 5, May 1986.

[12] Carey, M., Livny, M., and H. Lu, ”Dynamic Task Allocation in a Distributed Database System,” Proceedings of the 5th International Conference on Dis- tributed Computer Systems, Denver, May 1985.

[13] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, “Supporting Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems,” IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 4, pp. 580–593, Apr. 2011

[14] Hadoop Distributed File System, http://hadoop.apache.org/hdfs/.