Rebalancing the Chunks for Distributed File Systems in Clouds

Anila A[1*] Prof P Mohamed Shameem [2*]
1*PG Scholar, Dept. of Computer Science and Engineering
[2*] Professor, Dept. of Computer Science and Engineering
TKM Institute of Technology, Kollam, India

***Abstract*— Cloud computing is a commonly used term for the delivery of on demand computing resources over the internet. Distributed file system is a client/server based application where the server can be distributed across several physical computer nodes. Cloud computing application is based on the MapReduce programming is used in distributed file system. In such a file system each file is partitioned into several fragments called chunks and these chunks are distributed on the chunk servers. In a large scale cloud computing environment nodes are dynamically created, deleted, replaced and updated, this modifications causes load imbalance in distributed file system. To overcome the load imbalance problem in distributed file system, a completely distributed load rebalancing algorithm is presented. The load rebalancing task is used to eliminate the load on the central node. Using this algorithm the load is balanced as well as the migration cost, algorithmic overhead and load imbalance factor is reduced.**

Index Terms—Distributed file systems, Load balance, cloud computing, Chunks, Chunk server.

I. INTRODUCTION

Cloud computing is an emerging infrastructure in which number of computers are connected using communication network. One of the key issues in cloud computing is load balancing. In a distributed system the load balancing improves the system performance and resource utility. A distributed file system (DFS) is classical model developed mainly for large scale data storage and data access. Cloud computing application is based on the MapReduce programming [1] used in distributed file system. In Hadoop MapReduce paradigm is master slave architecture. The Master act like name node and Slave act like data node. Master takes large problem, divides it into the sub problem and assigns it to worker node. Both name node and data node are capable of computation and storage. The data node in the distributed file system stores the file chunks and the name node provides accesses like delete, create, open, append etc...Large scale distributed file system contains three parts:

- Main Server
- Chunk Server
- Client

Main server manages the file metadata, Chunk serve is used to manage the file storage of each node in the distributed file system and the client provides services by sending commands, as shown in Fig.1 below.

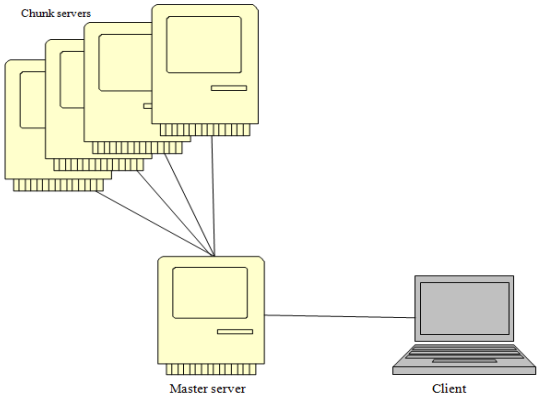

Fig 1 Large scale distributed file system

In a Distributed file system, a large file is partitioned into number of units called chunks .A chunk is a fragment of information, typically 64-128MB block size, and allocates each chunk to separate node called chunk server to perform MapReduce function parallel over each node.

In a large scale distributed file system Load balancing is the main issue. Load should be balanced over multiple nodes to improve system performance of the system, utilization of resource, response time and stability. In a distributed file system files can be upgraded, deleted, inserted and replaced, thus some of the files may be overloaded and under loaded. This results load imbalance in the distributed file system. The load rebalance is done by migrate the chunks from chunk servers of heavy load to light ones.

In this paper, a load rebalancing algorithm is introduced to overcome the load imbalance problem among the chunk servers in the distributed file system. The rebalancing task is done by migrating the chunks. The chunk servers in the distributed file system are divided into clusters according to the geographical locations in order to reduce the migration cost, and a prediction mechanism is used to predict the job arrival rate.

II. RELATED WORKS

Cloud computing has many challenging issues one among them is load balancing. Cloud cluster technique that divide cloud environment into number of clusters which help for the process of balancing the load [2].

Large-scale distributed file system may vary under certain conditions. File system is partitioned into single fragment of units called chunks and these chunks are allocated to the chunk servers [3]. The chunk servers may be added or deleted. This result load imbalance in the file system [4].For rebalancing the load in the distributed file system requires a great effort. Chunk migration is used to balance the load, for large files, migrates chunks from heavy load to light ones and for small files [5], it copies from heavy load to light loads.

Resources can be efficiently utilized and performance can be maximized in a load balanced cloud. In a centralized cloud computing system, if the central node fails performance bottleneck occurs and this results the failure of the whole system [6].

The storage nodes in the system are distributed as a network based on distributed hash tables [7]. Distributed Hash Tables are key building block for varieties of distributed applications in which discovering a file chunk can be refer to a key lookup mechanism. The DHT provides a unique identifier which is assigned to each file chunk. Because of the node arrival and departure the chunk servers in the DHT are self-configure and self-heal.

The chunk server in the distributed file system implements a DHT protocol such as chord [8] and [9]. A file is partitioned into a number of fixed-sized units called chunks, and each chunk has a unique identifier. To discover a file chunk in the chunk server DHT lookup operation is performed.

In the load rebalancing algorithm each node implements a protocol called gossip-based aggregation protocol [10] and [11], for collecting the load statuses of the randomly selected nodes in the system. Using this protocol each node in the system exchange a vector with its neighbours, the vector consists of load status of the selected node, network address and ID of each entries.

III. SYSTEM ARCHITECTURE

Node failure is common in a large scale distributed file system. In such a system each file is partitioned into fixed-size disjoint units.

A.Chunk Creation

A file is divided into a number of fragments of units called chunks, allocated to distinct nodes. The main objective is to distribute the file chunk as uniform as possible among the chunk servers. The load of a node usually proportional to the amount of file chunks the node possesses. In a load balanced state, each chunk server in a large scale distributed file system host no more than ideal number of chunks.

B. Cluster Formation

The chunk servers in the distributed file system are grouped into clusters according to the geographical locations. In a large cloud contains thousands of nodes and these nodes are in different geographical locations. By grouping the chunk servers into clusters reduces the migration cost. A prediction mechanism is used to predict the job arrival rate, by setting or predicting the job arrival rate can avoid the periodic checking of gossip protocol. This can reduce the algorithmic over head in the distributed file system.

B.DHT Formulation

The storage nodes in the distributed file system are structured as a network based on distributed hash tables (DHTs).The DHT provides a handle or identifier to every file chunk. For discovering a file chunk the unique identifier can be used. In a DHT network it has the property that, if a node leaves its locally hosted chunks are migrated to its successor and if a node joins then the node allocate the chunks whose IDs immediately precede the joining node from its successor to manage.

C. Replica Management

In distributed file systems such as Google GFS and Hadoop HDFS a fixed number of replicas or copies for each file chunk in the chunk servers are maintained in to improve file availability when a node failures and departures are occurred. The load rebalancing algorithm does not treat replicas distinctly. Because of the random nature of the algorithm two or more replicas are placed in the identical node.

D. Load Rebalancing Problem

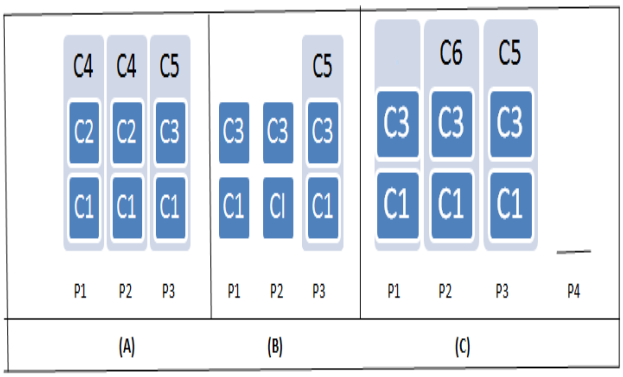

Fig 2 Load rebalancing problem

(A)In the example shows in three nodes P1, P2 and P3 the chunks are distributed.
(B) C4 is deleted from P1 and P2.
(C) Chunk C6 is appended and node P4 joins. The nodes in (B) and (C) are in load imbalanced state.

IV. LOAD BALANCING ALGORITHM

In the proposed algorithm, each chunk server in the large scale distributed file system first check the status of each node. Based on the amount of load, the nodes are divided into two types according to the amount of a load, they are

- Heavy node (Overloaded)
- Light node (Underloaded)

A node is heavy then it hosts the amount of chunk greater than a threshold of $(1+\Delta_H)$ *m* and if it is light the amount of chunk it hosts is less than the threshold of $(1-\Delta_U)$ *m*. Where Δ_H and Δ_U are used for representing the light and heavy nodes and *m* is ideal number of chunks hosted by a node.

The algorithm proceeds as follows, consider any node i∈*V*. If node *i* is the under loaded node in the system then it seeks an overloaded node. Node *i* leave system by migrating its locally hosted chunk into its successor and then rejoin as the successor of the heavy node. The light node request chunks from the overloaded node. Using gossip aggression protocol each node each node build a vector containing randomly selected nodes. Calculating the average load each node exchange its vector with its neighbours, the light node sort the node in the vector based on the load status including itself.

A large scale distributed file is in load balanced state, each node in the system host no more than the ideal number of chunks hosted by a node. This process of balancing repeats until all the heavy nodes in the systems become light or balanced.

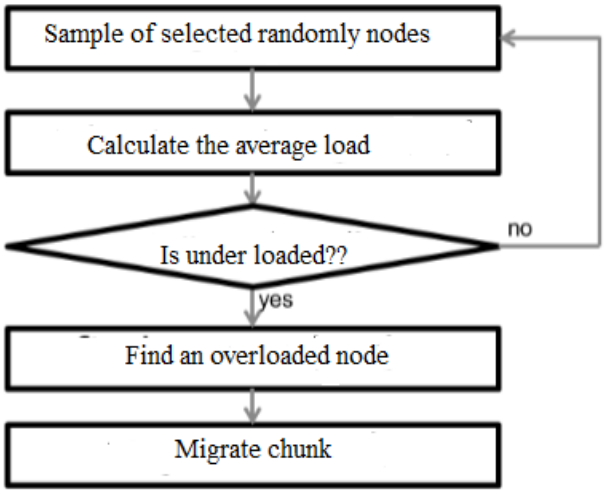

Fig 3 Algorithmic flow

Algorithm 1: A light node seeks an overloaded node.

Input: a vector 'V' contain sample of randomly selected nodes.
Output: an over loaded or heavy node.

(1) Begin
(2) Calculate the average load of nodes in the vector.
(3) If the load of a node is less than the threshold value, find itself as a light node then,
- Sort the nodes in its vector including the light node in ascending order based on the load status.
- The light node finds an overloaded node from the sorted vector.

Algorithm 2: For migration, a light node requests chunks from an overloaded node.

Input: a light node and an overloaded node.

(1) If the load of a node is greater than the threshold, the heavy node willing to share its load with light node then,
- The light node migrate its locally hosted chunks to its successor.
- Light node leaves the system.
- Rejoin the system as the successor of the heavy node.
- The heavy node allocates chunks to the light node to become balanced.

Hadoop cluster is a type of cluster designed for analyzing and storing large scale unstructured data in a distributed computing environment. In the system the chunk servers are grouped into clusters to reduce the migration cost. In the cluster load is balanced locally and globally at each time. Load balancing in the cluster the gossip protocol randomly select nodes.

Consider *c* is the total number of clusters and *α* is the amount of nodes randomly selected from the cluster for load balancing. Majority of the nodes are selected from the local cluster and some of the nodes are selected from the adjacent cluster. Each selected node implement gossip protocol and collect the load status, migration cost is added with the node selected from the adjacent cluster. A prediction is used for predicting the job arrival pattern.

V. PERFOMANCE EVALUATION

The proposed framework provides an efficient load rebalancing algorithm for balancing the load in a large scale distributed file system. The file chunks are distributed uniformly by reducing the migration cost. This algorithm also reduces the algorithmic overhead by using a prediction. The comparison with an existing distributed solution [12] which shows that the proposed system has less migration cost and algorithmic over head.

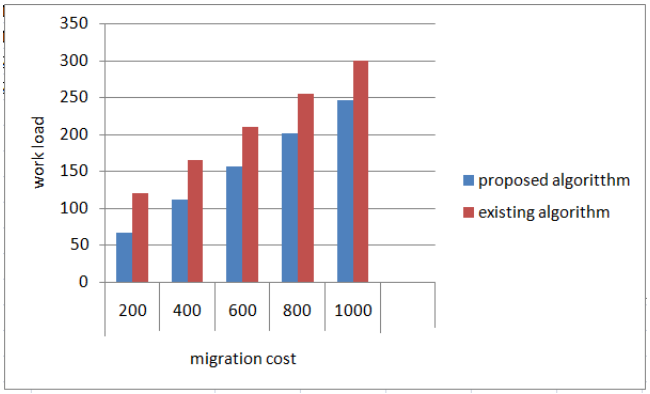

Fig 4 migration cost versus work load

A prediction is used to determine the job arrival rate. Predicting the job arrival rate can avoid the periodic checking of gossip algorithm. Comparison with the existing solution the proposed algorithm has less algorithmic overhead.

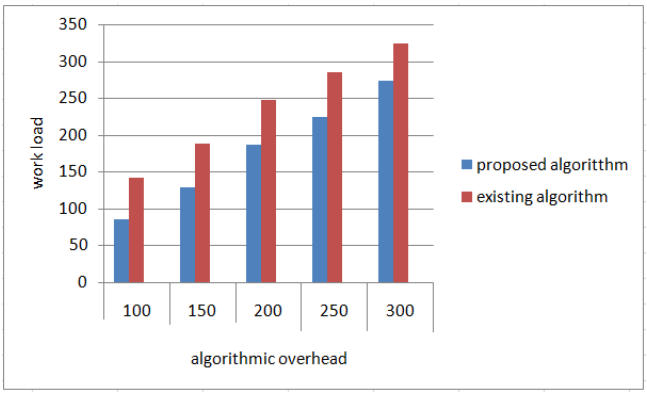

Fig 5 algorithmic overhead versus work load

VI. CONCLUSION

The core of a cloud computing infrastructure platform is a large scale distributed file system. For improving the overall performance of the system load balancing is one of the effective method. The load rebalancing algorithm effectively balances the load by distributing the file chunk uniformly as possible. The algorithm reduces the migration cost by clustering the chunk serves according to the geographical locations and the prediction mechanism avoided the periodic checking of gossip protocol thus reduces the algorithmic overhead.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Proc. Sixth Symp. Operating System Design and Implementation (OSDI '04), pp. 137-150, Dec. 2004.

[2] P.Jamuna and R.Anand Kumar "Optimized Cloud Computing echnique To Simplify Load Balancing" International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 11,November 2013.

[3]Hadoop Distributed File System "Rebalancing locks,"http://developer.yahoo.com/hadoop/tutorial/module2.html#rebalancing, 2012.

[4] V.Kalogeraki, P. M. Melliar-Smith, L. E. Moser. Dynamic migration algorithms for distributed object systems. The 21st Internationa Conference on Distributed Computing Systems, 2001:119-126.

[6] S.Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03), pp. 29-43, Oct. 2003.

[7] J.W. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '03), pp. 80-87, Feb. 2003.

[8] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," Proc.IFIP/ACM Int'l Conf. Distributed Systems Platforms Heidelberg,pp. 161-172, Nov. 2001.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek ,F. Dabek , and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," IEEE/ACM Trans. Networking, vol. 11, no. 1, pp. 17-21, Feb. 2003.

[10] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-Based Aggregation in Large Dynamic Networks," ACM Trans.Computer Systems, vol. 23, no. 3, pp. 219-252, Aug. 2005.

[11] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec , and M.V.Steen, "Gossip-Based Peer Sampling," ACM Trans. Computer Systems, v ol. 25, no. 3, Aug. 2007.

[12]Hung-Chang Hsiao, Hsueh-Yi Chung, HaiyingShen and Yu-Chang Chao, "Load rebalancing for distributed file systems in cloud" IEEE Trans. On parallel and distributed systems, vol. 24, no. 5, pp.951-962, May 2013