Mahesh.A *et al*, International Journal of Computer Science and Mobile Applications,
Vol.2 Issue. 9, September- 2014, pg. 15-20 **ISSN: 2321-8363**

# DISTRIBUTED FILE SYSTEM FOR LOAD REBALANCING IN CLOUD COMPUTING

**Mahesh.A[1], Pallavi.B[2], Shankar Thalla[3], Krishna Chaitanya Katkam[4]**
[1]M.Tech Student, CSE Dept., KCEA, Nizamabad, Telangana
Email-id: arepallymahesh@gmail.com
[2]Asst Prof., CSE Dept., KCEA, Nizamabad, Telangana
Email-id: varma.pallavi5@gmail.com
[3]Asst Prof., CSE Dept., KCEA, Nizamabad, Telangana
Email-id: shankar.thalla@gmail.com
[4]Asst Prof., CSE Dept., KCEA, Nizamabad, Telangana
Email-id: chaitanya.dynamics@gmail.com

**Abstract-**
Currently most of the cloud applications method great deal of knowledge to supply the required results in the information volumes to be processed by cloud applications is growing a lot of quicker than computing power. This difficulty in growth on new approaches for analyzing the knowledge and process. The employment of Hadoop Map Reduce framework to execute scientific workflows within the cloud this project explores. Cloud computing will providing the monumental clusters for economical giant division and information analysis. Distributed file system is simply a classical model of a file system used as the key building blocks for cloud computing. The above such files systems will partition a file into a number of chunks and allocated each chunk in to the distinct nodes. These Files are dynamically appended, created and deleted. The above result occurs load imbalance in a distributed file system; i.e., the file chunks are not uniformly distributed among the nodes. The distributed file systems in production systems strongly depend on a central node for chunk reallocation. To overcome this problem the distributed load rebalancing algorithm will be presented in this paper. To delete this dependence on the storage nodes each node performs the load rebalancing algorithm independently without acquiring global knowledge.

***Keywords****: Cloud computing, HDFS, Distributed hash tables, Load balancing*

## 1. Introduction

Cloud computing refers to applications and services offered over the Internet. This service is offered from data centers all over the world, these are collectively referred to as the "cloud". This metaphor is not having physical presence, yet universal nature of the Internet. The idea of the "cloud" simplifies the many network connections and computer systems involved in online services. Many network diagrams use the image of a cloud to represent the Internet. In fact, Cloud Computing is a technology and it connects so many nodes together for allocating resources dynamically. Different technologies are used in clouds such as distributed file systems, Map Reduce programming paradigm, and virtualization.

Distributed file systems are key building blocks for cloud computing applications based on the MapReduce programming paradigm. In such programming paradigm systems, nodes simultaneously serve computing and storage functions; a file is partitioned into a number of chunks allocated in distinct nodes so that MapReduce tasks can be performed in parallel over the nodes. In a distributed file system, the load of a node is proportional to the number of file chunks the node possesses. Because the files in a cloud can be created, deleted, and appended, and nodes can be upgraded, replaced and added arbitrarily in the file system, distribution of the file chunks are not uniformly distributed to among the nodes. Among storage nodes load balance is a critical function in clouds. State-of-the-art distributed file systems 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 design and implementation of a distributed file system simplified by the centralized approach. The recent experience concludes that when the number of files, the number of storage nodes and the number of accesses to files increase linearly, the performance of central nodes become a bottleneck, as they are unable to accommodate a large number of file accesses due to clients and MapReduce applications.

The storage nodes are structured as a network based on distributed hash tables (DHTs),[10] discovering a file chunk can simply refer to rapid key lookup in DHTs, given that a unique handle (or identifier) is assigned to each file chunk. DHTs enable nodes to self-organize and -repair while constantly offering lookup functionality in node dynamism, simplifying the system provision and management.

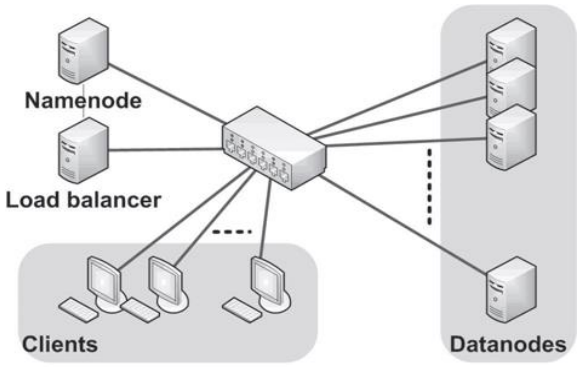


Fig 1: The experimental environment setup

## 2. Literature Survey

MapReduce [2] is a programming model and an associated implementation for processing and generating large data sets. A map function which is specified by user processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function which can merges all intermediate values associated with the same intermediate key. Most of the real world tasks are expressible in this model. The map and reduce primitives present in Lisp and many other functional languages. We learnt and realized that most of our computations involved applying a map operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and applying a reduce operation to all the values that shared the same key, in order that derived data can combine appropriately. The functional model with user specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

The Google File System [3] is scalable distributed file system for large distributed data-intensive applications. While running on inexpensive commodity hardware it provides fault tolerance and it delivers high aggregate performance to a large number of clients. The largest cluster to date provides hundreds of terabytes of storage across millions of disks on over a millions of machines, and it is concurrently accessed by thousands of clients. A GFS cluster consists of a single master and multiple chunk servers and is accessed by multiple clients. This includes the namespace, access control information, the current locations of chunks and the mapping from files to chunks. It also controls system-wide activities such as chunk lease management, garbage collection of orphaned chunks, and chunk migration between the chunk servers. The master can also communicate with each chunk server in HeartBeat messages to give it instructions and collect its state.

DHT based P2P systems offer a distributed hash table (DHT) abstraction for object storage and retrieval. Many solutions have been proposed to tackle the load balancing issue in DHT-based P2P systems [4]. But however, many solutions either ignore the reassign loads among nodes without considering proximity relationships or, heterogeneity nature of the system, or both. The goal is to ensure fair load distribution over

 

nodes proportional to their capacities, and also to minimize the load-balancing cost by transferring virtual servers between heavily loaded nodes and lightly loaded nodes in a proximity-aware fashion. There are two main advantages of a proximity-aware load balancing scheme. First and foremost, from the system perspective, a load balancing scheme bearing network proximity in mind can reduce the bandwidth consumption (e.g., bisection backbone bandwidth) dedicated to load movement. Second, it can avoid transferring loads across high-latency wide area links, thereby enabling fast convergence on the load balance and quick response to load imbalance.

A distributed peer-to-peer applications need to determine the node that stores a data item. The Chord [5] protocol solves this challenging problem in decentralized manner. Chord can provide support for just one operation: given a key, it maps the key onto a node. Chord simplifies the design of peer-to-peer systems and applications based on it by addressing these difficult problems:

Load balance**:** Chord acts as distributed hash function, spreading keys evenly over the nodes; this provides a degree of natural load balance.

Decentralization: Chord is fully distributed: no node is more important than any other. This improves robustness and as well makes Chord appropriate for loosely-organized peer-to-peer applications.

Scalability: The cost of a Chord lookup grows as the log of the number of nodes, so even very larger systems are feasible. No parameter tuning is required to achieve this scaling.

Availability: Chord automatically adjusts its internal tables to reflect node failures as well as newly joined nodes, ensuring that, the node responsible for a key can always be found, barring major failures in the underlying network. If the system is in a continuous state of change this will be true.

Flexible naming: Chord places no constraints on the structure of the keys it looks up: the Chord key-space is flat. This application gives a large amount of flexibility in how they map their own names to Chord keys.

A new framework, called Histogram-based Global Load Balancing (HiGLOB) [6] to facilitate global load balancing in structured P2P systems. Each node P in HiGLOB has two key components. The first component is a histogram manager that maintains a histogram that reflects a global view of the distribution of the load in the system. It is used to determine if a node is normally loaded, overloaded, or under loaded. The second component of the system is a load balancing manager that takes actions to redistribute the load whenever a node becomes overloaded or under loaded. The load-balancing manager may redistribute the load both statically when a new node joins the system and dynamically when an existing node in the system becomes overloaded or under loaded. We introduce two techniques that reduce the maintenance cost and reduce the cost of constructing histogram. Constructing a histogram for a new node may be expensive since it requires histogram information from all neighbor nodes. Additionally, the histograms of the new node's neighbors also need to be updated since adding a new node to a group of nodes changes the average load of that group. To partition the system into non-overlapping groups of nodes and maintain the average load of them in the histogram at a node. The reducing of overhead of maintaining and constructing histograms by the proposed techniques are used.

## 3. Load Rebalancing Algorithm

A node is light if the number of modules it hosts is smaller than the threshold as well as, a heavy node manages the number of modules greater than threshold. A large-scale distributed file system is in a load-balanced state if each module server hosts no more than A modules. In our proposed algorithm, each server node in module I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. This process repeats until all the heavy nodes in the system become light nodes. In Proposed system, file downloading or uploading with the help of the centralized system. Centralized system will be sharing the file (uploading and downloading). First of all we are going to notice the lightest node to require the set of modules from heaviest node. Thus we will do the method while not failure. Load equalization may be a technique to distribute employment across several computers or network to realize most utilization of resources economical output,

 

reducing latency, and take away overload. The load equalization service is sometimes provided by dedicated code or hardware, like a multilayer switch or name server. During this project we have a tendency to use Load rebalancing formula. Then identical method is dead to unleash the additional load on following heaviest node within the system. Then we are going to once more notice the heaviest and lightest nodes, such a method repeats iteratively till there's not the heaviest.

Advantages of Proposed System Using this we can use in large scale, failure-prone environment because the central load balancer is put under considerable workload that is linearly scaled with the system size. Another advantage of the proposed system is the security consistency provided by it. Various nodes with heavy loads have been proposed as alternatives to central node module so that so many drawbacks of the existing system can be avoided. The proposed system will help in keeping the system consistent so that we can avoid data loss.

Load Balancing Algorithm In our proposed algorithm, each module server node I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. If the number of modules it hosts is smaller than the threshold then a node is light. First we will find the lightest node to take the set of modules from heaviest node. So the process done without failure. To distribute workload across many computers or network to achieve maximum resource utilization, maximize throughput, minimize response time, and avoid overload by using of load balancing technique. The load equalization service is sometimes provided by dedicated software package or hardware, like a multilayer switch or name server.

Node Heterogeneity of Nodes participating in the file system are possibly heterogeneous in terms of the numbers of file chunks can accommodate that the nodes. we assumed that there is bottleneck resource by which we can get optimization although a node's capacity in is proportional to the bandwidth of the network, space & its computational power.

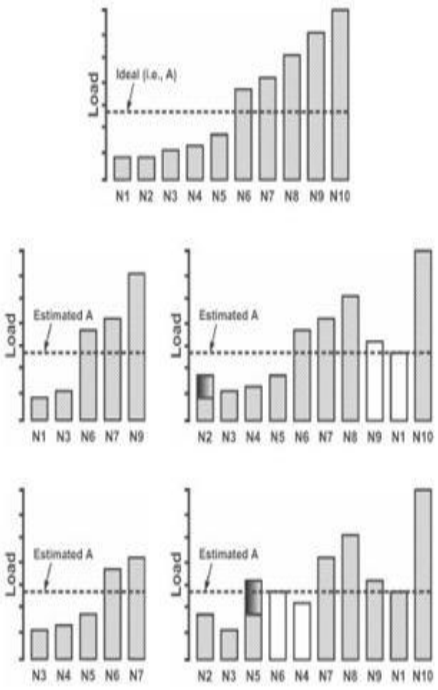


Fig.2. An example illustrating our algorithm, where (a) the initial loads of chunkservers N1; N2;..;N10, (b) N1 samples the loads of N1, N3, N6, N7, and N9 in order to perform the load rebalancing algorithm, (c) N1 leaves and sheds its loads to its successor N2, and then rejoins as N9's successor by allocating AeN1 chunks (the ideal

 

number of chunks N1 estimates to manage) from N9, (d) N4 collects its sample set fN3;N4;N5;N6;N7g, and (e) N4 departs and shifts its load to N5, and it then rejoins as the successor of N6 by allocating L6 _ AeN4 chunks from N6.

In Map reduced based Applications load is proportional to the no of file chunks in consideration of nodes. The rational design is to ensure that the number of file chunks managed by node i is proportional to its capacity. Replica Management (in Google GFS and Hadoop HDFS), a relentless variety of replicas for every file module square measure maintained in distinct nodes to enhance file handiness with relation to node failures and departures. Our current load equalization rule doesn't treat replicas clearly. It is unlikely that 2 or additional replicas square measure placed in a consistent node owing to the random nature of our load rebalancing rule. More specifically, every underneath loaded node samples variety of nodes, every elect with a likelihood of 1/n, to share their hundreds (where n is that the total variety of storage nodes).

## 4. CONCLUSION

In this paper we concluded that in large-scale, dynamic and distributed system having the drawback will be overcome by load equalization algorithm. Our proposal strives to balance the masses of nodes and scale back the demanded movement price the maximum amount as potential, whereas taking advantage of physical network vicinity and node no uniformity. Leave space for vendors to boost and optimize a completely unique load equalization algorithmic rule to modify the load-rebalancing drawback in cloud has been conferred during this paper. Best algorithmic rule is commonly topology specific.

## References

[1]. Hung-Chang Hsiao, Member, IEEE Computer Society, Hsueh-Yi Chung,Haiying Shen, Member, IEEE, and Yu-Chang Chao, proposed a "Load Rebalancing for Distributed File Systems in Clouds" *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 5, may 2013

[2]. 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.

[3]. 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.

[4]. Y. Zhu and Y. Hu, proposed a "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 4, pp. 349-361, Apr. 2005.

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

[6]. Q.H. Vu, B.C. Ooi, M. Rinard, and K.-L. Tan, proposed a "Histogram-Based Global Load Balancing in Structured Peer-to-Peer Systems," *IEEE Trans. Knowledge Data Eng.*, vol. 21, no. 4, pp. 595-608, Apr. 2009.

[7]. A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS '02), pp. 68-79, Feb. 2003.

[8]. Hadoop Distributed File System, http://hadoop.apache.org/hdfs/, 2012.

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

[10]. K. McKusick and S. Quinlan, "GFS: Evolution on Fast-Forward," *Comm. ACM, vol. 53, no. 3, pp. 42-49*, Jan. 2010.

[11]. HDFS Federation, http://hadoop.apache.org/common/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/Federation.html, 2012.

[12]. 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.

[13]. G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A.Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store,"*Proc. 21st ACM Symp. Operating Systems Principles (SOSP '07),*pp. 205-220, Oct. 2007.

[14]. D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," *Proc. 16th ACM Symp. Parallel Algorithms and Architectures (SPAA '04),* pp. 36-43, June 2004.

[15]. 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.

 

[16]. M. Raab and A. Steger, "Balls into Bins-A Simple and Tight Analysis*," Proc. Second Int'l Workshop Randomization and Approximation Techniques in Computer Science*, pp. 159-170, Oct. 1998.
[17]. 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.
[18]. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M.V. Steen, "Gossip-Based Peer Sampling," *ACM Trans. Computer Systems*, vol. 25, no. 3, Aug. 2007.
[19]. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A High Performance, Server-Centric Network Architecture for Modular Data Centers," Proc. *ACM SIGCOMM'09*, pp. 63-74, Aug. 2009.
[20]. H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic Routing in Future Data Centers," Proc. *ACM SIGCOMM* '10, pp. 51-62, Aug. 2010.
[21]. I. Raicu, I.T. Foster, and P. Beckman, "Making a Case for Distributed File Systems at Exascale," *Proc. Third Int'l Workshop Large-Scale System and Application Performance (LSAP '11),* pp. 11-18, June 2011.

## Authors' Biography

**Mahesh.A** received B.Tech from J.N.T.U, Hyderabad. He is an M.Tech(CS&E) Student in Kshatriya college of Engineering, Armoor. He is currently working for his M.Tech research project work.

**Pallavi.B** working as Assistant Professor in Kshatriya college of Engineering, Armoor. She received M.Tech from J.N.T.U, Hyderabad and she has 7 years of experience. Her interested subjects in Linux, C&DS and DBMS.

**Shankar Thalla** working as an Assistant Professor in Kshatriya College of Engineering, Armoor. He has completed his M.Tech CSE and he has 6 years of teaching experience. His research interested areas are Data Mining, Network Security and Cloud Computing.

**Krishna Chaitanya Katkam** working as an Asst. Professor in Kshatriya College of engineering, Armoor. He has completed his M.Tech CSE and he has 7 years of teaching experience. His research interested areas are Data Mining, Network Security and Cloud Computing.

 