Load Balancing Algorithm in Distributed File System: A Survey
(A Decision Making Strategy for Load Balancing in Cloud)

[1]Kalahasti K. P.
PG: student, Department of Computer Science and Engineering
RMD Engineering College
Kavaraipettai, Tamil Nadu, India

[2]Dr. Velvizhi N.
Professor, Department of Computer Science and Engineering
RMD Engineering College
Kavaraipettai, Tamil Nadu, India

Abstract— **Distributed File System is a key component in cloud computing application based on building block in Map Reduce Programming. In Distributed File System nodes are simultaneously serve computing and storage function and file are divided into number of chunk allocated in the distinct nodes In a large cloud we are able to add thousands of nodes together. The main aim of allocated files are not creating any significant load to any of the nodes, for the files are different partitioned squares measure off completely different modules. Another objective is to back the network traffic attribute and network inconsistencies to unbalancing hundreds of nodes. The result of reduction in network information measures in order than so man can run numerous such as large amount application run in it. Because of quantifiability property we are able to add, delete, update and new nodes in order that it supports heterogeneousness of the system. To enhance the potential of nodes we able tend to use Distributed file system in Cloud Computing Applications.**

Key Words—Cloud Computing, distributed Hash tables, load rebalancing

I. INTRODUCTION

Cloud Computing is a technology used for connecting so many nodes together for dynamically allocating resources. The Number of technology used in cloud such as virtualization,HdFS,Reduce programming paradigm, Distributed file System. These Kind of techniques scalable for add or del new node in system it making reliable. In a large cloud thousands of nodes are added together. the main aim of allocating files Distributed uniformly without making heavy load because of balance node to reduces the network traffic. The reduction of network inconsistency will lead to maximization of network bandwidth so that so many large applications can run in it. Due to scalability property can be added, deleted, updated new nodes so that it supports heterogeneity of the system. To improve the capability of nodes we use Distributed file System in Cloud Computing Applications. In such file systems the main functionalities of nodes is to serve computing and storage functions. If store a file into the system firstly we will divide the file into different modules and store it in different nodes. So introduced new load rebalancing algorithm to avoid all these disadvantages. When analyzing the existing system clouds rely on central nodes to balance the loads of storage nodes, they occur bottleneck because the failure of central nodes leads to the failure of whole system and it will leads to many technical and functional difficulties.

II. RELATED WORK

Google file System do not provide any caching technique in file system.GFS carried number of unreliable component and fault tolerance major problem in file system[1].some of problems occur by GFS operating system bugs, human error, disk failures etc.In File System once write are only read major drawback in the System. In new File System Hash Key values support constant number of load balancing method. It consist of two type schemes load stealing and load shedding the goal of scheme should me more cost effective and simple[2]. In extreme case ‘n’ balls and ‘n’ bins load balancing problem will occur in system. In File System Load balancing method even distribution of item should be limited. In DHT two type schemes present DHT address and DHT node[3]. Each node should be limited no of hash function and address should be balanced or limited no of address space. If address fails and node will fails this is major drawback in the System. In the design and evaluation of Pastry distributed object location and routing scheme for application of peer to peer network. Pastry not performs application-level routing and object location in a potentially very large overlay network of nodes connected via the Internet[4]. It can be used to support a wide range of peer-to-peer applications like global data storage, global data sharing, and naming. This paper Load balancing is necessary in such scenarios to eliminate skew. We presented asymptotically optimal online load-balancing algorithms that guarantee a constant imbalance ratio[5]. The data movement of cost per tuple insert or delete is constant, and was shown to be close to 1 in experiments. To adapt our algorithms to dynamic P2P environments, and architected a new P2P system that can be support efficient range queries.

This paper presents the design of Mercury, for supporting scalable protocol multi-nodes range-based searches. Mercury dicers from previous range-based query systems in that it supports multiple attributes as well as performs explicit load balancing[6]. To client routing and load balancing, Mercury uses light-weight sampling mechanisms novel for uniformly sampling random nodes in a highly dynamic overlay network. It shows that Mercury dicers not able to achieve its goals of logarithmic routing and uniform load balancing. Existing solutions to balance load in DHTs incur a high overhead either in terms of routing state or in terms of load movement generated by nodes incoming or outgoing the system[8]. This paper we have a tendency to propose a general techniques and use them to develop a protocol supported Chord, called Y0, that achieves load leveling with lowest overhead underneath the everyday assumption that the load is uniformly distributed within the symbol house. An elementary drawback that confronts peer-to-peer applications is that the economical location of the node that stores a desired information item. This paper presents Chord, a distributed operation protocol that addresses this drawback[9]. Chord provides only one operation: given a key, it maps the key onto a node. Information location is simply enforced on prime of Chord by associating a key with every information item, and storing the key/data combine at the node to that the key maps. Map Reduce may be a programming model AND an associated implementation for processing and generating massive information sets and users specify a map operate that processes a key/value combine to come up with a group of intermediate key/value pairs, and a cut back operate that merges all intermediate values related to constant intermediate key[10]. Several tasks are under able during this model, as shown within the paper. Programs written during this useful vogue are mechanically parallelized and dead on an oversized cluster of artifact machines.

III. PROPOSED SYSTEM

In proposed work the dependences will be eliminated on central nodes. the storage of files Distributed uniformly among the nodes with help of DHT.The DHT enables of nodes to repair and self-organize which offering constancy lookup based on functionality in the node Dynamism and providing system management. Our algorithm compared to existing approach. it provide production system and a competing distributed solution . The load rebalancing algorithm uniformly distributed the nodes equally without acquiring global knowledge.

A file is partitioned into a no of chunks allocation of nodes performed parallel over the Map Reduce task in the nodes. The load of a node is proportional typically to the number of file chunks the node possesses. Because the files are transform to the cloud can be automatically created, deleted, and update, and nodes can be added, replaced and upgrade in the file system, the file chunks are not uniformly distributed to the nodes.

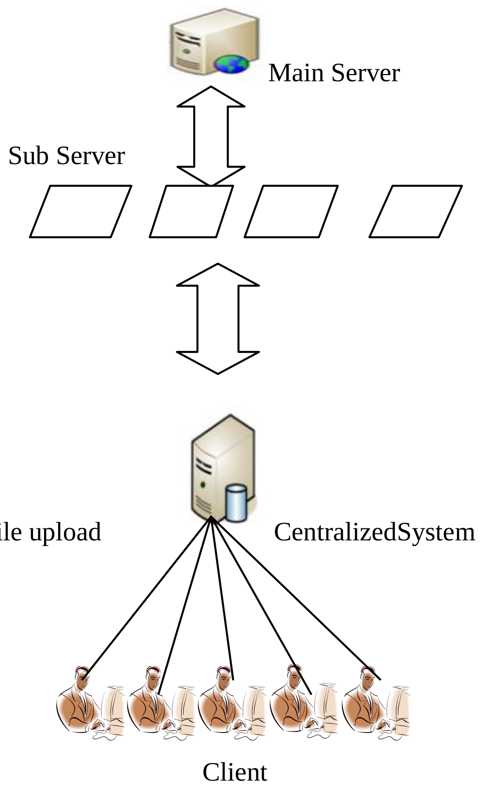

FIG.1 SYSTEM ARCHITECTUR

Our proposal is to allocate the number of chunk files distributed uniformly as possible among such that no can manage nodes excessive no of chunks.

The storage nodes are structured as a network based on distributed hash tables (DHTs), e.g., 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 carry out the nodes to self-organize and Repair constantly offering lookup functionality in node dynamism, simplifying the system management and provision. In our proposal the chunk server are organized as a DHT network. The Distributed Hash Table guarantee that if a node leaves, it will migrated to its successor node then if nodes leaves and joins the allocated chunks will manage ID immediately proceed the if a node joins and leaves then it allocates the chunks whose IDs immediately precede the node joining to its successor node manage it. our proposed algorithm, each chunk server node I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. If the node allocated no of chunks hosts is smaller than the threshold. Load statuses of a the nodes its selected each nodes randomly. Specifically, each node should select randomly in the system and builds a vector denoted by V. A vector consists of new entries, and each entry contains the ID and key value pair,(i.e)network address and load status of a randomly selected node.

IV.CONCLUSION

We should not standardize the algorithmic rule Inter-operability is not a problem devices implementing completely different algorithms will inter-operate. Load reconciliation Algorithms ought to be deterministic; a minimum of for a specific flow Frame order should be preserved at intervals a flow. The synthesis workloads across the load equalization algorithms by making a couple of storage node that square measure heavily loaded. In the Intermixture sensible algorithms continually works, given a standard receive algorithm. The potency and effectiveness of our style square measure valid by analytical models and a true implementation with a small-scale cluster setting. The centralized algorithmic rule within the Load rebalancing algorithm and dramatically outperforms the competitor distributed algorithmic rule in terms of load imbalance issue, movement of cost and avoid network traffic. Our proposal strives to balance the masses of nodes and scale back the demanded movement price the maximum amount as potential, where as taking advantage of physical network vicinity and node no uniformity.

V. FUTURE WORK

In future we have increase efficiency and effectiveness of our design are further validated by analytical models and a real implementation with a small-scale cluster environment highly desirable to improve the network efficiency by reducing each user's download time. In the contrast commonly focusing on average capacity in the network. It service involved capacity on both heterogeneity and the temporal correlation significantly increase download time in the network then average capacity of network will be remain same

VI.REFERENCE

[1] Sanjay Ghemawat and Howard Gobioff,, "The Google File System," SOSP'03 ,Bolton Landing, New York, USA Oct 2003.

[2] John Byers and Jeffrey Considine,"Simple Balancing for Distributed Hash Tables," SIGCOMM 2002.

[3] David R. Karger and Matthias Ruhl,"Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems" 2001.

[4] Antony Rowstron1 and Peter Druschel,"Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms Nov 2001.

[5] Prasanna Ganesan and Mayank Bawa,()"Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems" Proceedings of the 30th VLDB Conference,Toronto, Canada 2004.

[6] Ashwin R. Bharambe and Mukesh Agrawal,"Mercury: Supporting Scalable Multi-Attribute Range Queries," SIGCOMM'04,Portland, Oregon, USA sep 2004.

[7] Hung-Chang Hsiao,"Non-uniformity And Load Balance In Distributed Hash Tables" 7 Sept. 2009; revised 22 Dec. 2009; accepted 4 Apr. 2010;published online 18 May 2010.

[8] Ion Stoica and Robert Morris,"Chord: A Scalable Peer-To-Peer Operation Protocol For Web Application" SIGCOMM'01, San Diego, California, USA Aug 2001.

[9] Jeffrey Dean and Sanjay Ghemawat,"Map Reduce: Simplified On Massive Cluster," USENIX Association OSDI '04: 6th Symposium on Operating Systems Design and Implementation Feb 2004.