

# **Implementation of DHT on Load Rebalancing In Cloud Computing**

G.Naveen, J.Praveen Chander

PG Scholar, Department of CSE, Velammal Institute of Technology, Panchetti, Chennai, India

Assistant Professor, Department of CSE, Velammal Institute of Technology, Panchetti, Chennai, India

**ABSTRACT:** Distributed file systems are key building blocks for cloud computing applications based on the Map Reduce programming. In such file systems, nodes simultaneously serve computing and storage functions. Files can also be dynamically created, deleted, and appended. This results in load imbalance in a distributed file system. The file chunks are not distributed as uniformly as possible among the nodes. In existing we use round robin algorithm, due to this algorithm the load balance happens to the server good with some extend. All the servers should response in same time duration to complete the task.If any one of the server makes delay in response for the given task which impact the CPU computing resource. As a result it can be a bottleneck resource and a single point of failure. Our target is to optimize the computing resource (server), maximum throughput of servers, to avoid overload or crash of computing servers and to increase the response time. In our system DHT algorithm in such a way to make optimize computing resource and increase response time. In our model we divided the total bytes in to number of active servers and fed the same accordingly. This will make the effective utilization of the servers. And also we are divided the each file in to number of chunks for easy processing which increase the response time as well as easy error re-transmission if any data was dropped during transmission. Additionally, we aim to reduce network traffic or movement cost caused by rebalancing the loads of nodes as much as possible to maximize the network bandwidth available to normal applications.

**KEYWORDS:** Load balance, distributed file systems, clouds, Distributed file system

## **I. INTRODUCTION**

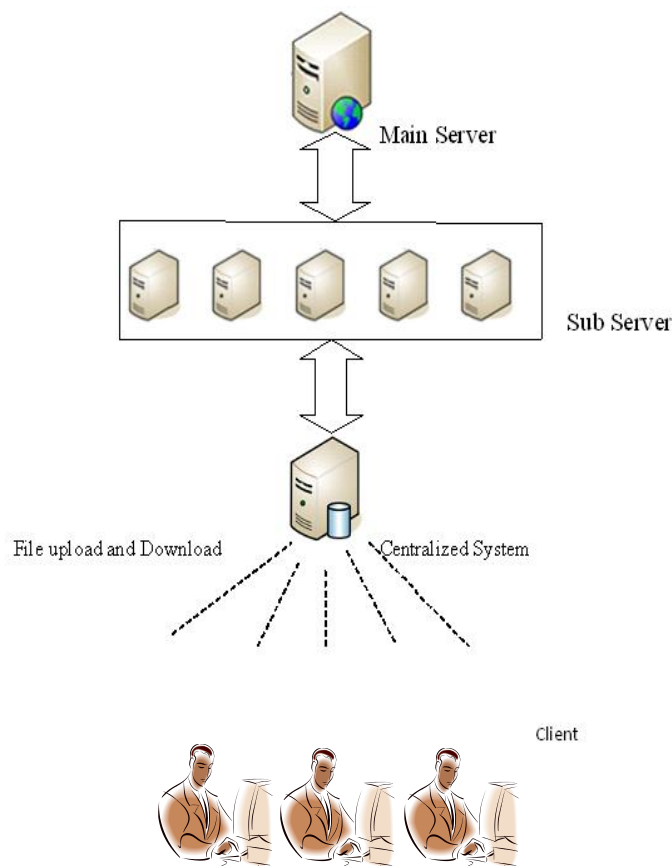
CLOUD Computing which refers to a distinct IT environment that is designed for the purpose of remotely provisioning scalable and measured in IT resources. Cloud computing becomes its own formalized IT industries segments, the symbols of clouds was commonly used to represent the Internet in a variety of specifications and mainstream documentation of Web-based architectures. In thecloud, the clients can dynamically allocate their resources on-demand without sophisticated deployment and management of resources. The Keys are enabling technologies for clouds include the MapReduce programming paradigm a way of building the structure and elements of computer programs and the distributed file systems are virtually used in cloud. These techniques emphasize scalability, so that clouds can be large network in scale, and comprising entities can arbitrarily fail and join while maintaining system reliability. The Distributed file systems are the key building blocks for cloud computing applications based on the MapReduce programming paradigm. In such file systems, nodes are 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. For example, let we consider a word count application that counts the number of distinct words and the frequency of each unique word in a larger files. In such an application, a cloud partitions the file into a large number of disjointed and fixed-size pieces (or file chunks) and assigns them to different cloud storage nodes (i.e., chunk servers). Each storage node (or node for short) then calculates the frequency of each unique word by scanning and parsing its local file chunks.

## II. RELATED WORKS

The leveraging DHTs, we present in load rebalancing algorithm for distributing file chunks as uniformly as possible and minimizing the movement cost as much as possible. We proposed a related algorithm which is operated on a distributed manner in which nodes perform their load balancing tasks independently without synchronization or global knowledge regarding to the system.

The Load balancing algorithms based on DHTs have been extensively studied. However, most existing system solutions are designed without considering both movement cost and node heterogeneity and may introduce significant maintenance network traffic to DHTs. In contrast, the proposal we made is not only takes advantage of physical network locality in the reallocation of file chunks to reduce the movement cost but also exploits capable nodes to improve the overall system performance.

Architecture diagram



Additionally, the algorithm we use it will reduces algorithmic overhead introduced to the DHTs as much as possible. Our proposals are assessed through computer simulations. The simulation wise the results are indicated to each node performs our load rebalancing algorithm independently without acquiring the global knowledge, our proposal will be comparable with the centralized approach in Hadoop HDFS and remarkably outperforms the competing distributed algorithm in terms of the load imbalance factors, movement cost and algorithmic overhead. So additionally, our load balancing algorithm are exhibits a fast convergence rate. We derive analytical models to the validation for the

efficiency and effectiveness of our design. Moreover, we have been implemented to our load balancing algorithm in HDFS and investigated its performance in a cluster environment.

### **III. EXISTING SYSTEM**

The popular file system for networked computers is the Network File System (NFS). It is a way to share files between machines on a network as if the files were located on the client's local hard drive. Frangipani is the scalable distributed file system that manages a collection of disks on multiple machines as a single shared pool of storage.

The machines are required to be under a common administrator and be able to communicate securely. The first one is that it depends on a single name node to manage almost all operations of every data block in the file system. As per the result it can be a bottleneck resource and a single point of failure.

There are some disadvantages in our existing system; it is a remote file system appears as a local file system. Compared to a local file system is not appropriate or reliable. It is a very simple internal structure which enables them to handle system recovery. Potential problem with HDFS is depends on TCP to transfer data.

### **IV. PROPOSED SYSTEM**

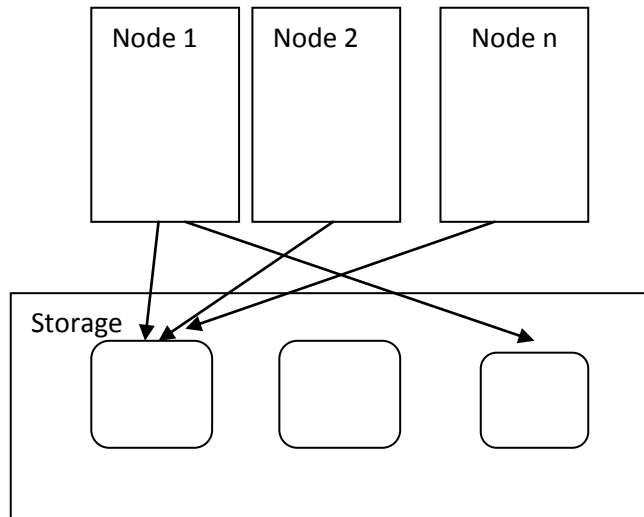
This eliminates the dependence on central nodes and the storage nodes are structured as a network based on distributed hash tables. DHTs enables the nodes to self organize and repair while constantly offering look up functionality in the node dynamism and which is simplifying the system provision and management. Our proposed algorithm is compared against a centralized approach in a production system and a competing distributed solution presented in the literature. The simulation which the results indicate that although each node performs our load rebalancing algorithm independently without acquiring global knowledge.

When DHT algorithm proposed the implementation proves some advantages, they are the load of each virtual server is stable over the timescale when load balancing is performed. We have implementation our load balancing algorithm in HDFS and investigated its performance in a cluster environment. Reduce the network traffic. The load rebalancing algorithm exhibits a fast convergence rate.

### **V. MODULES**

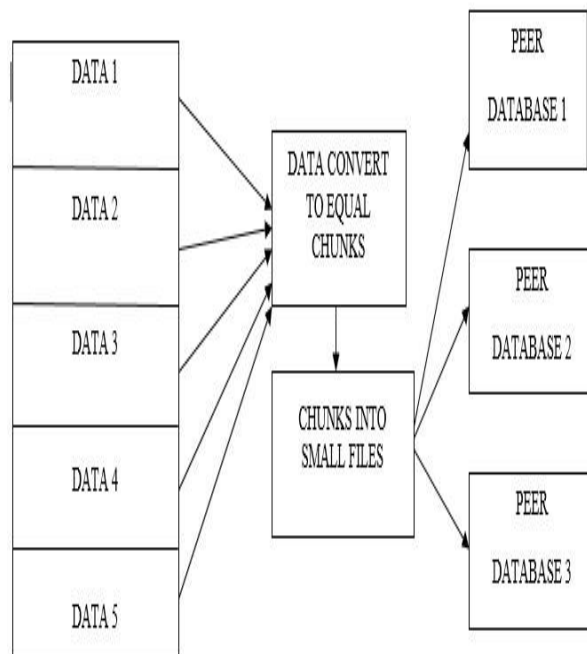
#### **CHUNK CREATION**

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. The load we present in a node is typically proportional to the number of file chunks the node possesses. The files in a cloud can be arbitrarily created, deleted, and modified and nodes can also be replaced and added in the file system, the file chunks are not distributed uniformly as much as possible among the nodes. Our main objective is to allocate the chunks of files as uniformly as possible among the nodes such that no node manages an excessive number of chunks.



**Chunk creation  
DHT FORMULATION**

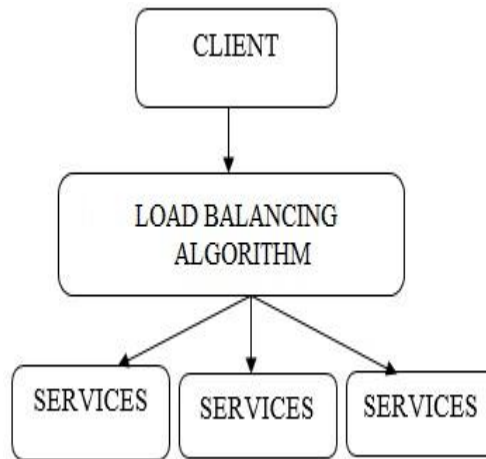
The storage nodes are structured as a network based on distributed hash tables (DHTs), e.g., discovering a file chunk can simply refer to rapid key lookup in DHTs, given that a unique handle (or identifier) is assigned to each file chunks. DHTs enable nodes to self organize and Repair while constantly offering look up functionality in node dynamism and its simplifying the system provision and management. The chunk servers in our proposal model are organized as a DHT network. If the typical DHTs guarantee that if a node leaves, then it is locally hosted file chunks are reliably migrated to its successor; if a node joins, then it allocates the chunks whose IDs are immediately precede the joining node from its successor to manage.



**LOAD BALANCING ALGORITHM**

In our proposed algorithm, each chunk server node I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. The nodes are light if the number of chunks it hosts is smaller than the threshold. The Load statuses of a sample of randomly selected nodes.

Specifically, each node contacts a number of randomly selected nodes in the system and builds a vector. A vector consists of entries, and each entry contains the ID, network address and load status of is randomly selected node.

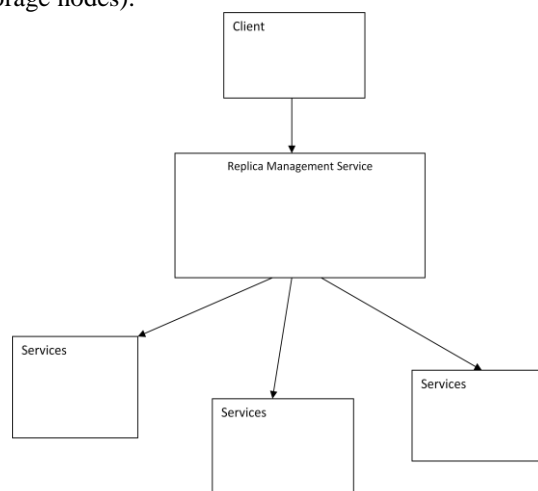


**Load Balancing Algorithm**

**REPLICA MANAGEMENT**

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 to improve file availability with respect to node failures and departures.

Our current load balancing algorithm does not treat replicas distinctly. It does unlikely that two or more replicas are placed in an identical node because of the random nature of our load rebalancing algorithm. Moreover specifically, each under loaded node samples a number of nodes, each of them selected with a probability of 1/n, to share their loads (where n is the total number of storage nodes).



# International Journal of Innovative Research in Science, Engineering and Technology

An ISO 3297: 2007 Certified Organization

Volume 4, Special Issue 2, February 2015

5<sup>th</sup> International Conference in Magna on Emerging Engineering Trends 2015 [ICMEET 2015]

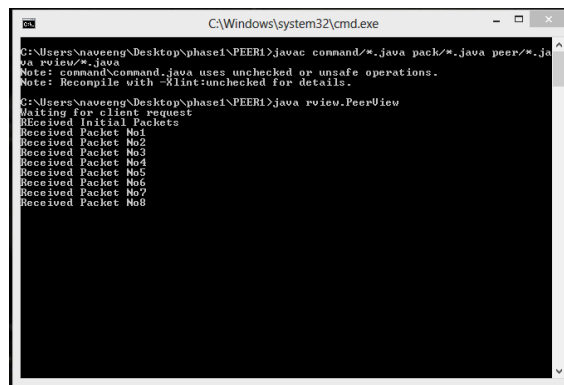
On 27<sup>th</sup> & 28<sup>th</sup> February, 2015

Organized by

Department of Mechanical Engineering, Magna College of Engineering, Chennai-600055, India.

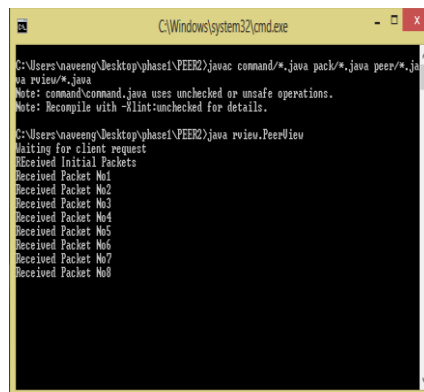
## Replica Management SCREENSHOTS

### CHUNK CREATION-PEER 1:



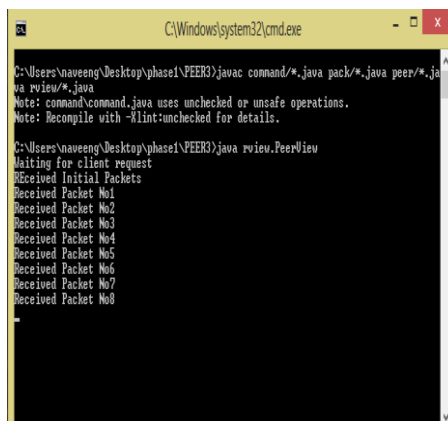
```
C:\Windows\system32\cmd.exe
C:\Users\Naveeng\Desktop\phase1\PEER1>javac command/*.java pack/*.java peer/*.java rview/*.java
Note: command\command.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
C:\Users\Naveeng\Desktop\phase1\PEER1>java rview.PeerUView
Waiting for client request
Received Initial Packets
Received Packet No1
Received Packet No2
Received Packet No3
Received Packet No4
Received Packet No5
Received Packet No6
Received Packet No7
Received Packet No8
```

### CHUNK CREATION-PEER 2:



```
C:\Windows\system32\cmd.exe
C:\Users\Naveeng\Desktop\phase1\PEER2>javac command/*.java pack/*.java peer/*.java rview/*.java
Note: command\command.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
C:\Users\Naveeng\Desktop\phase1\PEER2>java rview.PeerUView
Waiting for client request
Received Initial Packets
Received Packet No1
Received Packet No2
Received Packet No3
Received Packet No4
Received Packet No5
Received Packet No6
Received Packet No7
Received Packet No8
```

### CHUNK CREATION-PEER 3:



```
C:\Windows\system32\cmd.exe
C:\Users\Naveeng\Desktop\phase1\PEER3>javac command/*.java pack/*.java peer/*.java rview/*.java
Note: command\command.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
C:\Users\Naveeng\Desktop\phase1\PEER3>java rview.PeerUView
Waiting for client request
Received Initial Packets
Received Packet No1
Received Packet No2
Received Packet No3
Received Packet No4
Received Packet No5
Received Packet No6
Received Packet No7
Received Packet No8
```

### CHUNK CREATION-PEER 4:

**International Journal of Innovative Research in Science, Engineering and Technology**

An ISO 3297: 2007 Certified Organization

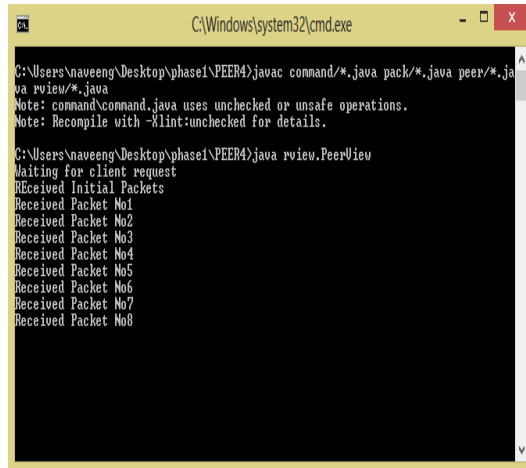
Volume 4, Special Issue 2, February 2015

**5<sup>th</sup> International Conference in Magna on Emerging Engineering Trends 2015 [ICMEET 2015]**

**On 27<sup>th</sup> & 28<sup>th</sup> February, 2015**

**Organized by**

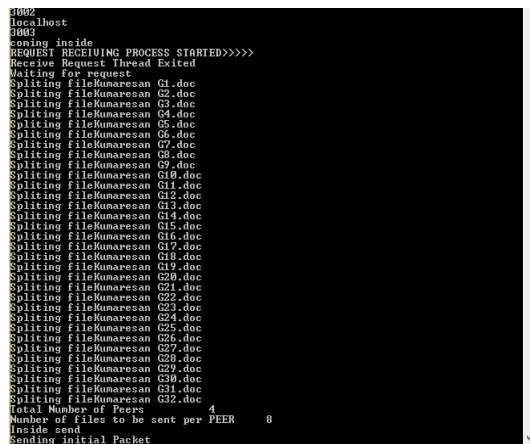
**Department of Mechanical Engineering, Magna College of Engineering, Chennai-600055, India.**



```
C:\Windows\system32\cmd.exe
C:\Users\ naveeng\Desktop\phase1\PEER4>javac command/*.java pack/*.java peer/*.java
va rview/*.java
Note: command\command.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\ naveeng\Desktop\phase1\PEER4>java rview.PeerView
Waiting for client request
REceived Initial Packets
Received Packet No1
Received Packet No2
Received Packet No3
Received Packet No4
Received Packet No5
Received Packet No6
Received Packet No7
Received Packet No8
```

**DHT FORMATION:**



```
3002
localhost
3003
coming inside
REQUEST RECEIVING PROCESS STARTED>>>>
Receive Request Thread Exited
Waiting for request
Splitting fileKunaresan G1.doc
Splitting fileKunaresan G2.doc
Splitting fileKunaresan G3.doc
Splitting fileKunaresan G4.doc
Splitting fileKunaresan G5.doc
Splitting fileKunaresan G6.doc
Splitting fileKunaresan G7.doc
Splitting fileKunaresan G8.doc
Splitting fileKunaresan G9.doc
Splitting fileKunaresan G10.doc
Splitting fileKunaresan G11.doc
Splitting fileKunaresan G12.doc
Splitting fileKunaresan G13.doc
Splitting fileKunaresan G14.doc
Splitting fileKunaresan G15.doc
Splitting fileKunaresan G16.doc
Splitting fileKunaresan G17.doc
Splitting fileKunaresan G18.doc
Splitting fileKunaresan G19.doc
Splitting fileKunaresan G20.doc
Splitting fileKunaresan G21.doc
Splitting fileKunaresan G22.doc
Splitting fileKunaresan G23.doc
Splitting fileKunaresan G24.doc
Splitting fileKunaresan G25.doc
Splitting fileKunaresan G26.doc
Splitting fileKunaresan G27.doc
Splitting fileKunaresan G28.doc
Splitting fileKunaresan G29.doc
Splitting fileKunaresan G30.doc
Splitting fileKunaresan G31.doc
Splitting fileKunaresan G32.doc
Total Number of Peers 4
Number of files to be sent per PEER 8
inside send
Sending initial Packet
```

**VI. CONCLUSION**

In this paper, our proposal strives to balance the loads of nodes and reduce the demanded movement cost are possible, while taking advantages of physical network locality and node heterogeneity. The absence of the representative real workloads (i.e., the distributions of file chunks in a large scale storage system) in the public domain, we have been investigated the performance of our proposal and compared it against competing algorithms through synthesized probabilistic distributions of the file chunks are emerging the distributed file systems in production systems strongly depend on a central node for chunk reallocation. Thus dependence are clearly inadequate in a large network scale and failure prone environment because the central load balancer is put under considerable workload that is linearly scaled with the system size and may become the performance bottleneck and the single point of failure. Our algorithm is compared against the centralized approach in a production system and a competing distributed solution presented in the literature. The simulation which results the indication that our proposal is comparable with the existing centralized approach and considerably outperforms the prior distributed algorithm in terms of load imbalance factors, movement cost, and algorithmic overhead. In this paper, we propose the fully distributed load rebalancing algorithm is presented to cope with the load imbalance problem.

## **VII. FUTURE ENHANCEMENT**

In future we have increase efficiency and effectiveness of our design is 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 contrast, the commonly held practice focusing on the motion of average capacity, we have been shown that the both spatial heterogeneity and the temporal correlations in the service capacity can significantly increase the average download time of the users in the network, even though when comparing the average capacity of the network remains the same.

## **REFERENCES**

- [1] J. Dean and S. Ghemawat, "Map Reduce: Simplified Data Processing on Large Clusters," Proc. Sixth Symp. Operating System Design and Implementation (OSDI '04), pp. 137-150, Dec. 2004.
- [2] S. Ghemawat, H. Gobi off, and S.-T. Leung, "The Google File System," Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03), pp. 29-43, Oct. 2003.
- [3] Hadoop Distributed File System, <http://hadoop.apache.org/hdfs/>, 2012.
- [4] VMware, <http://www.vmware.com/>, 2012.
- [5] Xen, <http://www.xen.org/>, 2012.
- [6] Apache Hadoop, <http://hadoop.apache.org/>, 2012.
- [7] Hadoop Distributed File System "Rebalancing Blocks," [http:// developer.yahoo.com/hadoop/tutorial/module2.html#rebalancing](http://developer.yahoo.com/hadoop/tutorial/module2.html#rebalancing), 2012.
- [8] K. McKusick and S. Quinlan, "GFS: Evolution on Fast-Forward," Comm. ACM, vol. 53, no. 3, pp. 42-49, Jan. 2010.
- [9] HDFS Federation, <http://hadoop.apache.org/common/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/Federation.html>, 2012.