The working of these two functions are, Map function is to produce a set of key pairs coming between in place or order, whereas Reduce function combines into a set of key pair of that values which is occurs together with coming between key values in place or order in a identical manner. The performance of parallelize operation is to easy and re-execute a system of parts in a machine for fault tolerance in the function of map and reduce. In the execution time, when system maintains their whole information. This is the division of input data into parts, a time table of an event of the program execution across number available machines, deal with failures and producing intercommunication between machines. The nodes are in parallel which performs to compute and the storing operation in the distributed file system. The large file is the division of chunks into a number of parts and assign it to definite nodes to make a MapReduce operation is parallel over nodes. Typically, the processing of task for MapReduce is on many terabytes of the information/data of much more different machines.

2.2 The Google File System [8][10]

The file system is scalable distributed file system for a great size of distributed data concentrated application. Implementation of the Google file system is based on a characteristic of supporting a great size of scalable distributed data processing workload on valuable hardware. The file system supplies fault tolerance by repeatedly testing the operation, very importance data is an exact copy, fast and automatic recovery and italso handover high performance which is a whole combines several elements of the great number of users. In this there are hundreds of terabytes of storing operations across millions of disk on over a millions of machines for their largest cluster also it is concurrently performance with the thousands of users. In a Google file system cluster having aindividual master and numerous chunk server and is performs a numerous users. The system includes the namespace, control information, the current location of chunks and the mapping from files tochunks. The control of the system which is an management of the chunk leases, unused collection of orphans to chunks. Chunks are move from one side and settle in another side which they between in the chunk servers. Each chunk servers are communicated between the master and the chunk server is place at the master message called as HeartBeat message which is passes the instruction and also bring its state.

2.3 Chord: A Scalable Peer-To-Peer Operation Protocol for Web Application

For web application Scalable Peer-To-Peer Operation is used. An elementary drawback that confronts peer-to-peer applications is that the economical location of the node that stores a desired information item. Drawbacks of peer-to-peer applications are to stored item desired information for economical location of the node. In this paper we overcome this drawback using distributed operation protocol. Chord provides support for single operation: set 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 combines at the node to that the key maps. Nodes are adopt by chord expeditiously and leave the system, and may answer queries even though the system is continuously dynamical. Results from simulations and theoretical analysis shows that Chord is scalable: communication price and therefore the state maintained by every node scale logarithmically with the amount of Chord nodes.

2.4 Load Balancing Algorithm for DHT Based Structured Peer to Peer System [14]

P2P system depends upon the DHT which offers abstraction for object storage and retrieval. Various solutions has been proposed for DHT based P2P system to tackle the load balancing issue. But on the other hand, several solutions either ignore the shift loads among nodes without considering heterogeneity nature of the system, proximity relationships or, both. The aim is to make sure even load distribution over nodes proportional to their capacities, and transferring virtual servers between heavily loaded nodes and lightly loaded nodes in a proximity-aware fashion for minimize the load-balancing cost. a proximity-aware load balancing scheme having the two main advantages and they are, from system viewpoint ,can reduce the bandwidth consumption for load balancing scheme dedicated to load movement. Another it can stay away from transferring loads across high latency wide area links, thereby quick response to load imbalance and enabling fast convergence on the load balance.

2.5 Histogram-Based Global Load Balancing in Structured Peer-to-Peer Systems[9]

A new era, called Histogram-based Global Load Balancing (HiGLOB) to assist global load balancing in structured P2P systems. for each node P in HiGLOB there are two key components. the first component is The histogram manager , it conserves a histogram that replicates global view of the distribution of the load in the system. It is used to determine if a node is normally overloaded, loaded, or under loaded. The load balancing administrator is second component of the system, which takes arrangements to reallocate the load whenever a node becomes overloaded or under loaded. The load-balancing administrator may reallocate the load both statically when a new node links the system and dynamically when an surviving node in the system becomes overloaded or under loaded. Here author presented two techniques that decrease the preservation cost and decrease the cost of histogram creation. Creating ahistogram for a new node may be more costly since it requireshistogram information from all its neighbour nodes. Moreover,the histograms of the new node’s neighbours also essential to beupdated since count a new node to a group of nodes deviationsthe regular load of that group. To barrier the system into nonoverlappinggroups of nodes and preserve the average load ofthem in the histogram at a node. The dropping of overhead ofpreserving and creating histograms by the planned techniquesare used.

3. Existing System

Distributed file system like Google GFS [8] and Hadoop HDFSin clouds depend on central nodes to control the metadatain order of the file system and to balance the loads

ofstorage nodes based on that metadata. The centralized approachsimplifies the design and implementation of a distributed filesystem. But, recently we examine that when the number ofaccesses, the number of files and the number of storage nodesto files increase linearly, the central nodes (e.g., the masterin Google GFS) become a performance bottleneck, as they areincapable to put up a large number of file accesses due to MapReduce applications. The module servers may be replaced orupgraded and added in the system because of the node failure.Let F is the set of files, Files in F may be randomly created,appended and deleted. In view of a large-scale distributedfile system containing a bunch of module servers M in acloud. Each file f has fixed-size modules and partitioned intoa number of disjointed. Surviving results to balance load inDHTs incur a high overhead either in terms of load or interms of routing state movement generated by nodes arrivingor departing the system. All DHTs create some effort to loadbalance, usually by(i) making each DHT node responsible for a stable portion ofthe DHT address space (ii) randomizing the DHT address associated with each item with a good enough hash function. First, the typical random partition of the address space among nodes is not completely balanced. a larger portion of arbitrarily distributed items are received because some nodes are end up with a large portion of the addresses. The even distribution of items is the key problem with DHTs. so as a result is uneven distribution of module servers [14].

A. Drawbacks of Existing System

- High Movement Cost
- High Network Traffic
- Algorithm Overhead
- Load Imbalance
- Relying on Central Node
- Security difficulties

4. Proposed System: Load Rebalanced Architecture

In our proposed system 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. In Figure 2. Our proposed system we will be able to rebalancing of node dynamically. And the most important part is security that we can provide while rebalancing of load in distributed file system. The security features in CDH4 enable Hadoop to prevent malicious user impersonation. And in end-to-end system we are going to used MD5 encryption algorithm for data security.

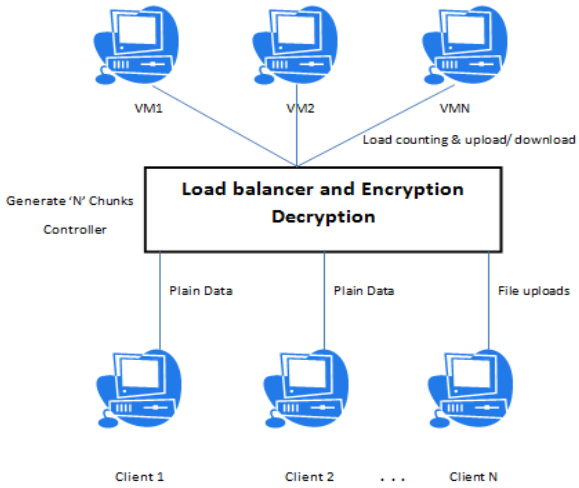


Figure 2: Load Rebalanced System Architecture

Advantages of Proposed System

- Deploy wide-range and failure error domain
- Reduce Network Traffic or Movement Cost
- Maximize the Network Bandwidth
- Improve overall System Performance
- Utilizes Physical Network Locality
- Better throughput and response time
- System consistency (i.e., avoid data loss)
- Excellent security
- Picking lead of node heterogeneity
- Extends resource utilization

5. Algorithms

Algorithm: LoadCalculation(Vid)

Input : provide VM id as Vid

Step 1: user select VM id from available VM's
Step 2: calculate CPU time and memory allocation by VM
Step 3: calculate load on performance as VL
Step 4: return VL

Output : CPU load in %

Algorithm: Chunk creation

Input: Text files from user Ti

Step 1: user selects Ti randomly
Step 2: initialize [ ] TotServers for all server id's
Step 3: for (i : available server sid)
TotServers[i]=sid[i]
Step 4: calculate cpuLoad[] each server
LoadCalculation(sid[i])
End for
Step 5: create file chunks base on reserve server space
ChunkServer();
Step 6: allocate each chunk to specified server.
Step 7: if (Transaction)
Return file store successfully
Else
Error in file upload

Output: save chunk on servers

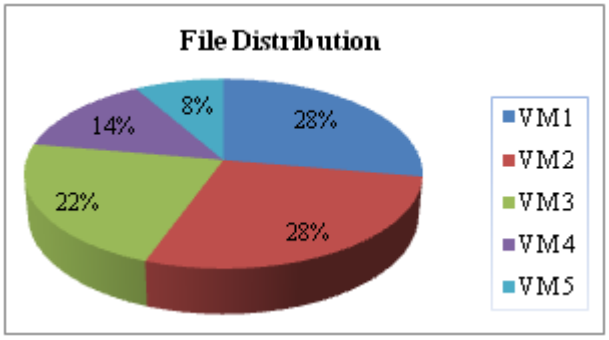
Procedure Chunk Server ()

Step 1: Read File and calculate filesize
Step 2: Repeat step 2 from 1 to noServers.
Step 3: Distribute file content using bytesPerSplit= (CpuLoad[sid]/100) * filesize.
Step 4: Write File into ChunkServer.
Step 5: Repeat step 2 (noOfServer)
Step 6: End Function

6. Results

Table 1: Load Distribution Results

VM Name	CPU Utilization	File Chunk Size	
VM 1	50 %	2.8 MB	5.6 MB
VM 2	50 %	2.8 MB	5.6 MB
VM 3	60 %	2.2 MB	4.4 MB
VM 4	75 %	1.4 MB	2.8 MB
VM 5	85 %	0.8 MB	1.6 MB
	Total Size	10.0 MB	20 MB



7. Conclusion

Load rebalancing algorithm is demonstrated to cope with the load imbalance problem. Transfer the file data separated different parts of files using load rebalancing and data encryption stranded DES algorithm after store up into clouds. Dynamically rebalancing of nodes present in a system which are overloaded or under loaded and minimizing the movement cost as much as possible. In proposed approaches by taking advantage of physical network locality and node heterogeneity can be done efficiently. Secured Load Rebalancing for Distributed File Systems in Private Clouds i.e. our proposed system can discard the issue like high delays, handle heterogeneous resources, efficiently adjust to dynamic operational conditions, offer efficient task distribution, and so it can provide minimum node idle time. Emerging distributed file systems in production systems strongly depend on a central node for chunk reallocation. This dependence is clearly incompetent in a large-scale, failure-prone environment because the central load balancer is put under considerable workload that is linearly scaled withthe system size, and possibly it will become the performance bottleneck and the single point of failure.

8. Future Scope

In future we can also implement the file type as jpeg, mp3, mp4 etc. Using same Load Rebalancing approach we can implement for images, audio, video etc. with better security.

References

[1] 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
[2] HDFSFederation, http://hadoop.apache.org/common/docs/r0.23.0/hadoop-yarn/hadoop-yarn site/Federation.html, 2012. cPGCON 2015, MET's Institute of Engineering
[3] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica,"Load Balancing in Structured P2P Systems," Proc. SecondInt'lWorkshop Peer-to-Peer Systems (IPTPS '02), pp. 68-79, Feb. 2003.
[4] J. Dean and S. Ghemawat, "Map Reduce: Simplified Data Processing on Large Clusters," in Proc. 6th Symp. Operating System Design and Implementation (OSDI'04), Dec. 2004, pp. 137–150.
[5] Hadoop Distributed File System, http://hadoop.apache.org/hdfs/.
[6] Hadoop Distributed File System, "Rebalancing Blocks,"http://developer.yahoo.com/hadoop/tutorial/module2.htm lrebalancing
[7] J. W. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," in Proc. 1st Int'l Workshop Peerto- Peer Systems (IPTPS'03), Feb. 2003, pp. 80– 87.
[8] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in Proc. 19th ACM Symp. Operating Systems Principles (SOSP'03), Oct. 2003, pp. 29–43.
[9] 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.
[10] K. McKusick and S. Quinlan, "GFS: Evolution on Fast-Forward," Comm. ACM, vol. 53, no. 3, pp. 42-49, Jan. 2010.
[11] 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.
[12] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems,"Performance Evaluation, vol. 63, no. 6, pp. 217-240, Mar. 2006.
[13] P. Ganesan, M. Bawa, and H. Garcia-Molina, "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems, "Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04),pp. 444-455, Sept. 2004.
[14] ChahitaTanak, Rajesh Bharati "Load Balancing Algorithm for DHT Based Structured Peer to PeerSystem"International Journal of Emerging Technology and Advanced Engineering (ISSN 2250-2459, ISO9001:2008 Certified Journal, Volume 3, Issue 1, January 2013)

Author Profile

Mr. Jayesh Kamble is a student of Masters in Engineering, Computer Department, PVPIT ,Pune University. He received Bachelors of Engineering in 2013 from Pune University. His research interests are Computer Networks (Software Defined Networks), Network Security, Distributed Systems etc.

Prof. Y. B. Gurav is working as Associate Professor and Head of Department of Computer Engineering in PVPIT, Pune University with 16 years of teaching experience. He Completed his master in engineering (CSE) And now he is perusing his Ph.D. His research interests are Network Security, Distributed Systems, HCI, Compiler systems etc.