International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Design of a High Performing Cloud Using Load Rebalancing Technique in Distributed File System

Y. Steeven[1], C. Prakasha Rao[2]

[1]M. Tech Student, Prakasam Engineering College, Kandukur, Prakasam (Dt), India

[2]M.Tech, [Ph.D] Associate Professor, CSE Department, Prakasam Engineering College, Kandukur, Prakasam (Dt), India

Abstract: ***Distributed file systems are key building blocks for cloud computing applications based on the Map Reduce programming paradigm. In such file 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. However, in a cloud computing environment, failure is the norm, and nodes may be upgraded, replaced, and added in the system. Files can also be dynamically created, deleted, and appended. This results in load imbalance in a distributed file system; that is, the file chunks are not distributed as uniformly as possible among the nodes. Emerging distributed file systems in production systems strongly depend on a central node for chunk reallocation. This dependence is clearly inadequate in a large-scale, failure-prone environment because the central load balancer is put under considerable workload that is linearly scaled with the system size, and may thus become the performance bottleneck and the single point of failure. In this paper, a fully distributed load rebalancing algorithm is presented to cope with the load imbalance problem. Our algorithm is compared against a centralized approach in a production system and a competing distributed solution presented in the literature. The simulation results indicate that our proposal is comparable with the existing centralized approach and considerably outperforms the prior distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead. The performance of our proposal implemented in the Hadoop distributed file system is further investigated in a cluster environment.***

Keywords: Cloud Computing, Load Rebalancing, Distributed File System, Movement Cost, Network Traffic

1. Introduction

Cloud Computing (or cloud for short) is a compelling technology. In clouds, clients can dynamically allocate their resources on-demand without sophisticated deployment and management of resources. Key enabling technologies for clouds include the Map Reduce programming paradigm [1], distributed file systems, virtualization, and so forth. These techniques emphasize scalability, so clouds can be large in scale, and comprising entities can arbitrarily fail and join while maintaining system reliability. Distributed file systems are key building blocks for cloud computing applications based on the Map Reduce programming paradigm. In such file systems, nodes simultaneously serve computing and storage functions; a file is partitioned in to a number of chunks allocated in distinct nodes so that Map Reduce tasks can be performed in parallel over the nodes.

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. In such a distributed file system, the load of a node is typically proportional to the number of file chunks the node possesses. Because the files in a cloud can be arbitrarily created, deleted, and appended, and nodes can be upgraded, replaced and added in the file system, the file chunks are not distributed as uniformly as possible among the nodes. Load balance among storage nodes is a critical function in clouds. In a load-balanced cloud, the resources can be well utilized and provisioned, maximizing the performance of Map Reduce-based applications. State-of-the-art distributed file systems (e.g., Google GFS[7],[8] and Hadoop HDFS [3]) 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 centralized approach simplifies the design and implementation of a distributed file system. However, recent experience concludes that when the number of storage nodes, the number of files and the number of accesses to files increase linearly, the central nodes (e.g., the master in Google GFS) become a performance bottleneck, as they are unable to accommodate a large number of file accesses due to clients and Map Reduce applications. In this paper, we are interested in studying the load rebalancing problem in distributed file systems specialized for large-scale, dynamic and data-intensive clouds. (The terms “rebalance” and “balance” are interchangeable in this paper.) Such a large-scale cloud has hundreds or thousands of nodes (and may reach tens of thousands in the future).

Our 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. 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. Moreover, as failure is the norm, nodes are newly added to sustain the overall system performance, resulting in the heterogeneity of nodes. Exploiting capable nodes to improve the system performance is, thus, demanded. Specifically, in this study, we suggest offloading the load rebalancing task to storage nodes by having the storage nodes balance their loads spontaneously. This eliminates the dependence on central nodes.

A. Virtualization:

Virtualization is the most profound change that PCs and servers have experienced, said Simon Crosby, chief technology officer for Citrix Systems’ Data Center and Cloud Division [9]. “IT departments have long been at the mercy of the technical demands of legacy applications", explained Chris Van Dyke, [10] Microsoft's chief technology strategist for the oil and gas industry. "Now, rather than having to maintain older operating systems because of the needs of a legacy application, IT departments can take advantage of the performance and security gains in a new OS (in one virtual machine) while supporting legacy applications in another. Also, the process of deploying applications becomes simpler, because applications can be virtualized and deployed as a single virtual machine". [14]Virtualization technology lets a single PC or server simultaneously run multiple operating systems or multiple sessions of a single OS. This lets users put numerous applications even those that run on different operating systems on a single PC or server instead of having to host them on separate machines as in the past. The approach is thus becoming a common way for businesses and individuals to optimize their hardware usage by maximizing the number and kinds of jobs a single CPU can handle.[13].

B. Hyperviser: IaaS software is low-level code that runs independent of an operating system called a hypervisor , and isresponsible for taking inventory of hardware resources and allocating resources based on demand.[12]

C. Private Cloud: The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

D. Parellel Data Processing: Particular tasks of processing a job can be assigned to different types of virtual machines which are automatically instantiated and terminated during the job execution, parallel.

E. Distributed File System: Files are stored on different storage resources, but appear to users as they are put on a single location. A distributed file system should be transparent, fault-tolerant and scalable.

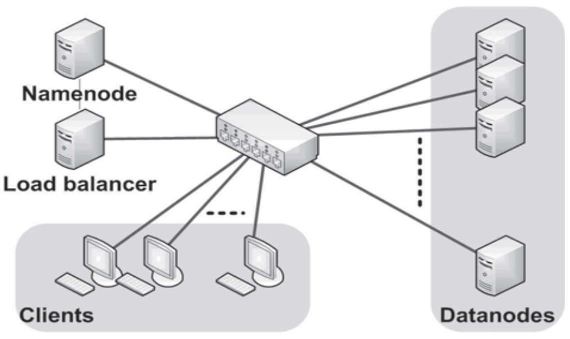


Figure 1: The experimental environment setup

2. Literature Survey

Map Reduce [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.

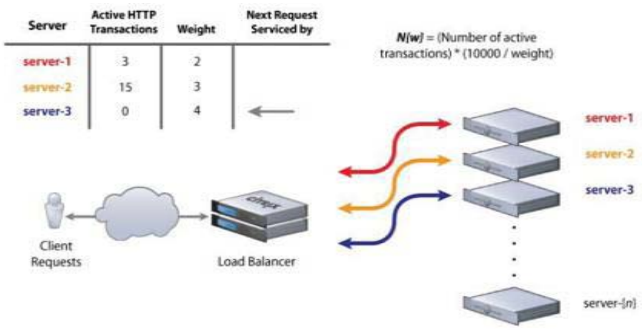
A distributed peer-to-peer applications need to determine the node that stores a data item. The Chord [5] protocol solves this challenging problem in decentralized manner. Chord can provide support for just one operation: given a key, it maps the key onto a node. Chord simplifies the design of peer-to-peer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

a) **Load balance:** Chord acts as distributed hash function, spreading keys evenly over the nodes; this provides a degree of natural load balance.
b) **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.
c) **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.
d) **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.
e) **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 Balancing Algorithm

In our projected algorithm, each chunk server would firstly estimate whether the nodeis under loaded (light) or overloaded (heavy) without global knowledge. A node is said to be light if the number of chunks it hosts is smaller than the threshold value. The load status sample of randomly selected nodes is given below.



Server	Active HTTP Transactions	Weight	Next Request Serviced by
server-1	3	2	
server-2	15	3	
server-3	0	4	←

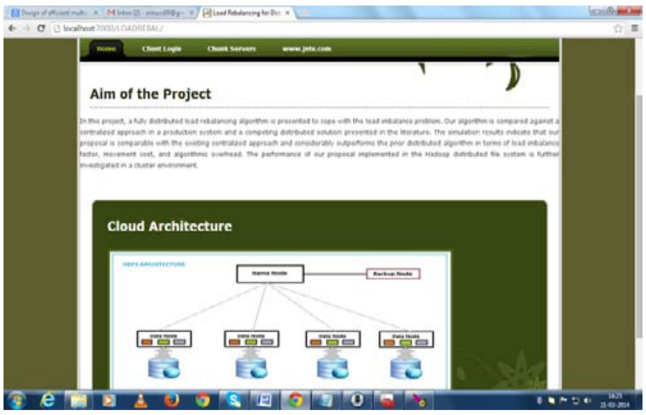
Figure 2: Load balancing

3.1 Load-Balanced State

If each one of the chunk server do not host not more than ‗Am‘ chunks. In our projected algorithm, each chunk server node ‗I‘ firstly estimate whether it is under loaded(light) or overloaded (heavy) exclusive of global knowledge. ‗ ‘ of ‗A‘ from ‗j‘ is used to relieve the load of ‗j‘ node ‗j‘ may possibly still remain as the heaviest node in the system after it has migrated its load to node ‗i‘. In such cases, the current least-loaded node, say node‗I‘ departs and then rejoins the system as a successor of ‗j‘. That is the new node ‗I‘ becomes node ‗j+1‘, and j‘s original successor ‗i‘ thus becomes node ‗j + 2‘. Such a process repeats iteratively until ‗j‘ is no longer the heaviest. Then, the same process is executed to release the extra load on the next heaviest node in the system. This process repeats until all the heavy nodes in the system become light nodes. We will offer a rigorous performance analysis for the effect of varying in Appendix E. Specifically; we discuss the tradeoff between the value of and the movement cost. A larger introduces more overhead for message exchanges, but results in a smaller movement cost.

Procedure 1 ADJUSTLOAD (Node Ni) fOn Tuple Insertg
```
1: Let $L(N_i) = x\ 2\ (T_m ; T_m +1]$.
2: Let $N_j$ be the lighter loaded of $N_i$ -1 and $N_i$ +1.
3: if $L(N_j)\ T_m$ _ 1 thenf DoNBRADJUSTg
4: Move tuples from $N_i$ to $N_j$ to equalize load.
5: ADJUSTLOAD($N_j$)
6: ADJUSTLOAD($N_i$ )
7: else
8: Find the least-loaded node Nk.
9: if $L(N_k)$ _ $T_m$ +2then fDoREORDERg
10: Transfer all data from $N_k$ to N = $N_k$ _1.
11: Transfer data from Ni to $N_k$, s.t. $L(N_i) = d_x$=2e and $L(N_k) = b_x$ =2c.
12: ADJUSTLOAD (N)
13: fRename nodes appropriately after REORDER.g
14: end if
15: end if
```

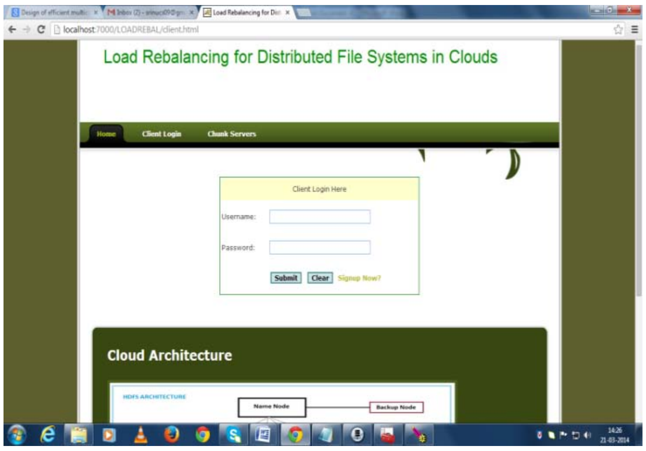
4. Simulation Results



Aim of the Project

In this project, a fully distributed load rebalancing algorithm is presented to cope with the load imbalance problem. Our algorithm is compared against a centralized approach in a production system and a competing distributed solution presented in the literature. The simulation results indicate that our proposal is comparable with the existing centralized approach and considerably outperforms the prior distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead. The performance of our proposal implemented in the Hadoop distributed file system is further investigated in a cluster environment.

Figure: home page

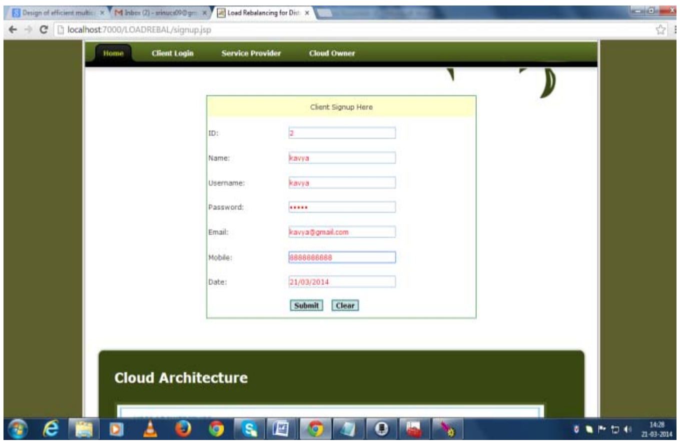


Client Login Here

Username:

Password:

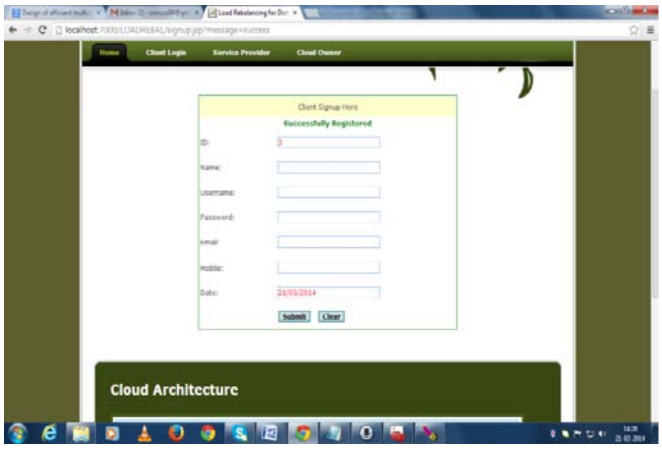
Figure: client login pag



Client Signup Here

ID:	2
Name:	kavya
Username:	kavya
Password:	•••••
Email:	kavya@gmail.com
Mobile:	8888888888
Date:	21/03/2014

Figure: Client signup pag

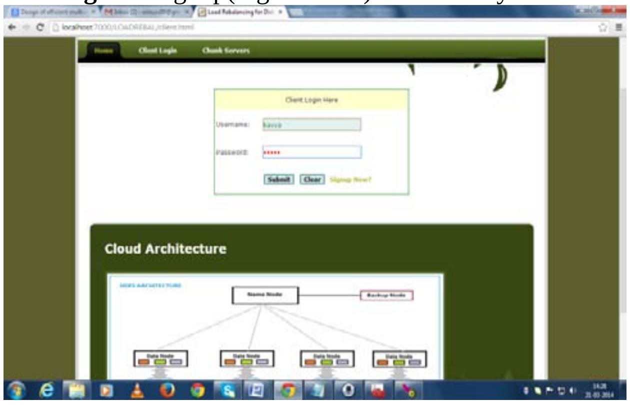


Client Signup Here

Successfully Registered

ID:	3
Name:	
Username:	
Password:	
email:	
mobile:	
Date:	21/03/2014

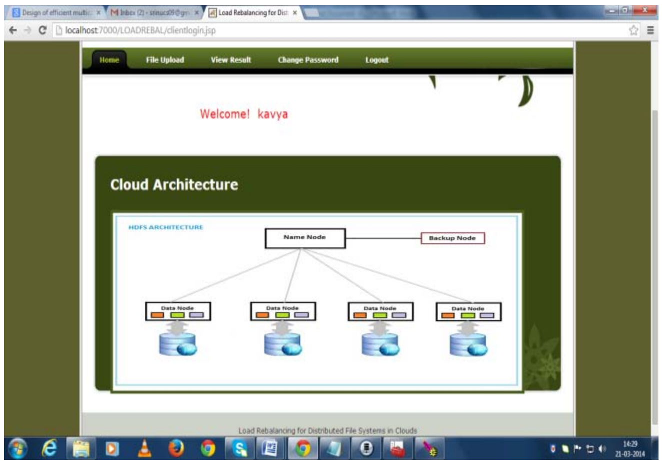
Figure: Signup(registration) successfully done



Client Login Here

Username:	kavya
Password:	•••••

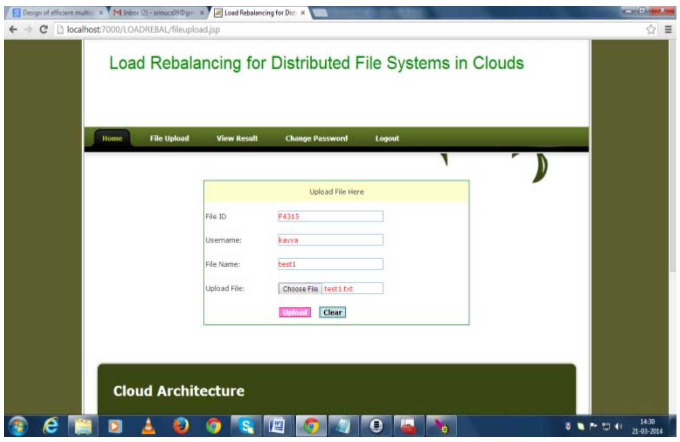
Figure: regestred client or user login pag



Welcome! kavya

Load Rebalancing for Distributed File Systems in Clouds

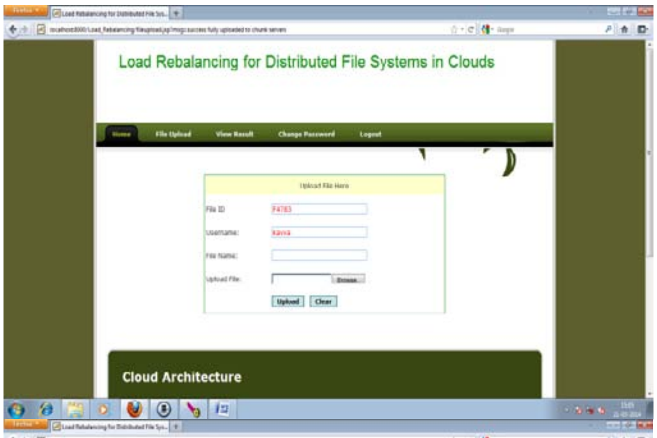
Figure: client login in to the accout(say)


localhost:7000/LOADREBAL/fileupload.jsp
Load Rebalancing for Distributed File Systems in Clouds
Home
File Upload
View Result
Change Password
Logout
Choose File
Upload
Clear
Cloud Architecture

Upload File Here

File ID	F4315
Username:	kavya
File Name:	test1
Upload File:	test1.txt

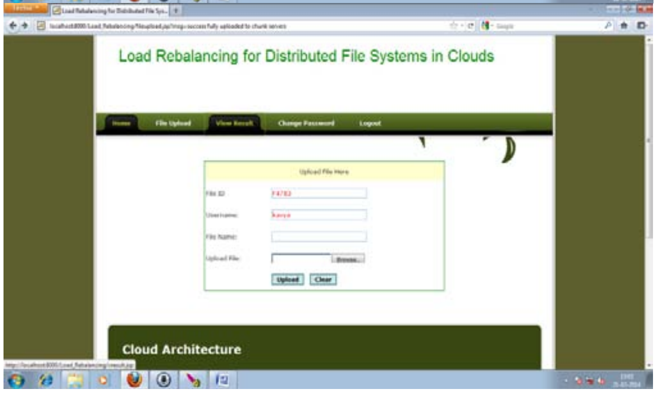
Figure: file uploading file



Upload File Here

File ID	F4783
Username:	kavya
File Name:	
Upload File:	Browse...

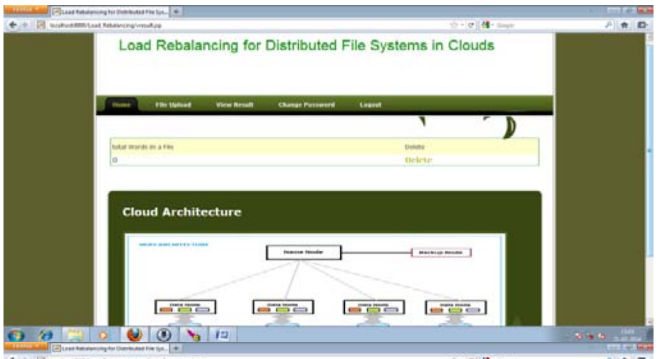
Upload Clear



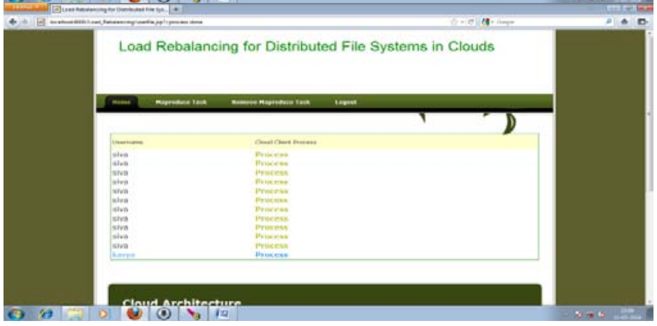
Upload File Here

File ID	F4783
Username:	kavya
File Name:	
Upload File:	Browse...

Upload Clear

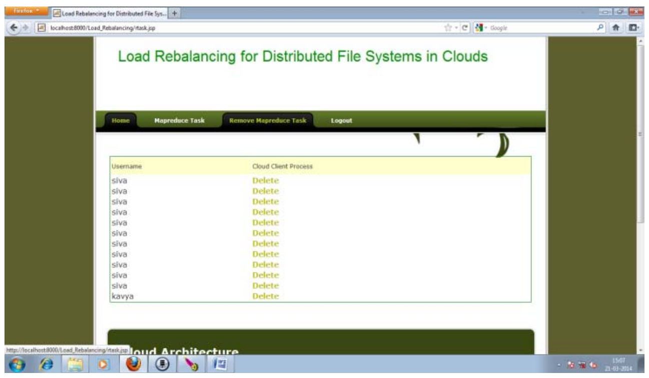


Total Words in a File	Delete
0	Delete



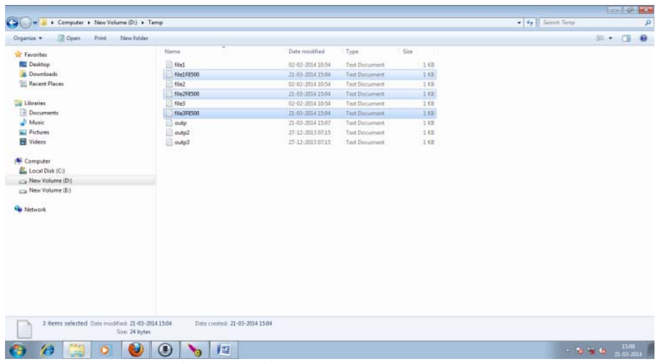
Username	Cloud Client Process
siva	Process
siva	Process
siva	Process
siva	Process
siva	Process
siva	Process
siva	Process
siva	Process
siva	Process
siva	Process
siva	Process
kavya	Process

Figure: clint under processs



Username	Cloud Client Process
siva	Delete
siva	Delete
siva	Delete
siva	Delete
siva	Delete
siva	Delete
siva	Delete
siva	Delete
siva	Delete
siva	Delete
siva	Delete
kavya	Delete

Figure: clinet deleted or removed by admin or server



Name	Date modified	Type	Size
file1	02-02-2014 10:54	Text Document	1 KB
file1F8500	21-03-2014 15:04	Text Document	1 KB
file2	02-02-2014 10:54	Text Document	1 KB
file2F8500	21-03-2014 15:04	Text Document	1 KB
file3	02-02-2014 10:54	Text Document	1 KB
file3F8500	21-03-2014 15:04	Text Document	1 KB
outp	21-03-2014 15:07	Text Document	1 KB
outp2	27-12-2013 07:15	Text Document	1 KB
outp3	27-12-2013 07:15	Text Document	1 KB

3 items selected Date modified: 21-03-2014 15:04 Date created: 21-03-2014 15:04
Size: 24 bytes

Figure: file(s) after portioned or splited (say)

5. 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] 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] 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.
[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, 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.
[10] 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.
[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] 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.
[13] 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.
[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] 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.
[16] 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.
[17] G.S. Manku, “Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables,” Proc. 23rd ACM Symp. Principles Distributed Computing (PODC ’04), pp. 197-205, July 2004.
[18] A. Bharambe, M. Agrawal, and S. Seshan, “Mercury: Supporting Scalable Multi-Attribute Range Queries,” Proc. ACM SIGCOMM ’04, pp. 353-366, Aug. 2004.
[19] Y. Zhu and Y. Hu, “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.
[20] H. Shen and C.-Z. Xu, “Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured P2P Networks,” IEEE Trans. Parallel and Distributed Systems, vol. 18, no. 6, pp. 849-862, June 2007.
[21] Q.H. Vu, B.C. Ooi, M. Rinard, and K.-L. Tan, “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.
[22] H.-C. Hsiao, H. Liao, S.-S. Chen, and K.-C. Huang, “Load Balance with Imperfect Information in Structured Peer-to-Peer Systems,” IEEE Trans. Parallel Distributed Systems, vol. 22, no. 4, pp. 634-649, Apr. 2011.
[23] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., 1979.
[24] D. Eastlake and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” RFC 3174, Sept. 2001.
[25] 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.
[26] 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.
[27] 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.