RESEARCH ARTICLE OPEN ACCESS

Efficient load rebalancing for distributed file system in Clouds

Mr. Mohan S. Deshmukh*, Prof. S.A.Itkar**

(Department Of Computer Engineering. Pune University, ME II Year P.E.S's Modern College Of Engineering)
*** (Department Of Computer Engineering, Pune University, Assistant Professor P.E.S's Modern College Of*

ABSTRACT
Cloud computing is an upcoming era in software industry. It's a very vast and developing technology. Distributed file systems play an important role in cloud computing applications based on map reduce techniques. While making use of distributed file systems for cloud computing, nodes serves computing and storage functions at the same time. Given file is divided into small parts to use map reduce algorithms in parallel. But the problem lies here since in cloud computing nodes may be added, deleted or modified any time and also operations on files may be done dynamically. This causes the unequal load distribution of load among the nodes which leads to load imbalance problem in distributed file system. Newly developed distributed file system mostly depends upon central node for load distribution but this method is not helpful in large-scale and where chances of failure are more. Use of central node for load distribution creates a problem of single point dependency and chances of performance of bottleneck are more. As well as issues like movement cost and network traffic caused due to migration of nodes and file chunks need to be resolved. So we are proposing algorithm which will overcome all these problems and helps to achieve uniform load distribution efficiently. To verify the feasibility and efficiency of our algorithm we will be using simulation setup and compare our algorithm with existing techniques for the factors like load imbalance factor, movement cost and network traffic.
***Keywords* -** Clouds, Distributed File system, Load balance, Movement cost, Network traffic

I. INTRODUCTION

Cloud computing is advanced technology in which dynamic allocation of resources on-requirement basis is carried out without any specific procedure. Cloud computing is scalable since it uses key technologies like MapReduce algorithms [4], distributed file systems [2], [8], virtualization etc. Distributed file systems are usually used for cloud computing, which are based on MapReduce technology. In this file system, files are divided into number of small parts and these parts are allocated to various distinct nodes. Nodes serve both storage and computing functions. For example, consider an application where counting a number of distinct names of the persons in a given country. Then application will find out distinct names of the person and also the frequency of each name. In this type of application, a cloud partitions the file into fixed size parts and then assigns them to different

nodes in the system. Then each node will perform the counting task on part of file stored in it. But as discussed the nodes may be upgraded, deleted or added dynamically and also the file chunks. This leads to load imbalance problem. Uniform load distribution is challenging task in cloud computing. In a load balanced environment performs of the system will improve and we can achieve high efficiency.

For load balancing mostly used approach is central node technique [3]. In this technique dependency is on central node for managing metadata information of the file systems for balancing loads of storage nodes. This approach is useful to simplify the design and implementation of the distributed file system [2]. But the main concern for this approach is scalability of cloud computing system. As the number of nodes, number of files and users accessing the system increases central load fails due to overload and performance bottleneck problems. New developments were carried out to solve these problems in central node but those techniques do not serve the purpose successfully. For example, in Hadoop DFS [8] federation architecture with multiple namenodes for managing metadata information is used. In this system manual and static portioning is carried out [9]. But since the load of namenodes may change over a period of time and there is no provision for migration of load for load balancing, any namenode with excess load may become a bottleneck and node without any load will be ideal at the same time. So this technique also fails to give uniformly load balance system.

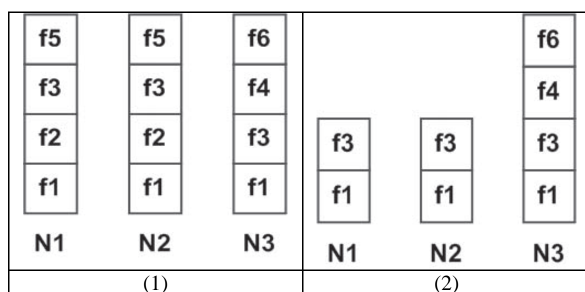
In this paper, we emphasizes on studying load rebalance problem in distributed file systems which are large-scale, dynamic and data intensive. This type of large-scale file systems has hundreds or thousands of nodes. Our main objective is to

design a system which will have uniform load distribution as far as possible. We also focus on movement cost i.e. the network traffic caused by nodes in interest of balancing the loads and also improving the capacity of nodes which will enhance the performance of the overall system. We mainly suggest the offloading the load rebalancing work to storage nodes by availing the storage nodes to balance their loads spontaneously. For this we structure the nodes in DHT network format. Assignment of unique chunk handle for each file chunk (part) is carried out. DHTs will allow nodes to self-organize and repair by offering lookup function in node.

In this paper will cover the following points. 1st will go through the load rebalancing problem, then our proposed algorithm for load rebalancing and then evaluation of algorithm through computer simulation and assessment of simulation result. Then using cluster environment performance measurement of our proposal.

II. WHAT IS LOAD REBALANCING PROBLEM?

1. Consider a large-scale distributed file system [8] which consists of set of chunk servers V in a cloud and cardinality of V is n. n can be ten thousand or more. Files will be stored on n chunk servers. Let's say set of files as F. each file f belongs to F is partitioned into number of parts and fixed-size chunks denoted by Cf .Second load of chunk servers is directly proportional to number of chunks hosted by the server. Chunk servers may be replaced, added or upgraded in the system and also the files F may be appended, created or deleted at any time [2]. This in turn affects the load balancing of system and results in non-uniform load distribution in the system. Figure 1 illustrates an example of load rebalancing problem. Assumption here is that chunk servers are homogenous and have same capacity.



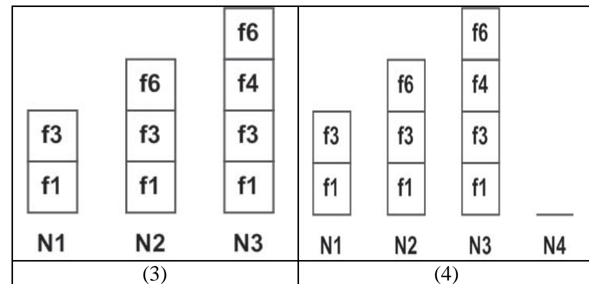


Fig. 1 example of load rebalancing problem, (1) initial load distribution, (2) files f2 andf5 are deleted (3) f6 is appended and (4) node 4 joins. The nodes 1, 2, 3 are in load imbalanced state.

We will focus on designing a load rebalancing algorithm to reallocate the chunks to achieve the uniform load distribution to the system as much as possible. We will also try to reduce the movement cost i.e. the migration of chunks caused for balancing loads of chunkservers [3]. Let A be the ideal number of chunks that any chunkserver i belongs to V is required to manage in load balanced state,

$$A = \frac{\sum_{f \epsilon F} |C}{n}$$

Then we aim to minimize the load imbalance factor in each chunkserver i as follows: ||Wi- .A||

(2) Where Wi is load of node and $\|\cdot\|$ denotes the absolute value function

The load rebalancing problem is NP- hard. 1st for simplicity we will assume a homogenous environment, where migration of file chunk between any two nodes requires a unit movement cost and each chunk server has the identical storage capacity. But for practical considerations we will have to deal with nodes with heterogeneous capacity and different movement costs.

III. SYSTEM ARCHITECTURE

Organization of chunk servers as DHT network i.e. implementation of each chunk server using DHT protocols Chord and Pastry. Partitioning of files into number of fixed-size chunks, and each chunk will have a unique chunk identifier known as chunk handle (SHA1). This hash function returns a unique chunk identifier for a given file's path name string and chunk index. For example, the identifiers of 1st and 10th chunks of file "/user/sap/tmp/Y.log" are respectively SHA1 (/user/sap/tmp/Y.log,0)and SHA1 (/user/sap/tmp/Y.log, 10). Each chunk server with unique ID will represented as 1, 2, 3…n. Successor of chunk server will be i+1 and successor of chunk server n as chunk server 1. To discover a file chunk, the DHT lookup operation is performed.

3.1 Adavantages Of Using Dhts

By making use of DHTs, it guarantees that if any node leaves, then its locally hosted chunks are reliably migrates to its successor and if node joins, then it allocates the chunks who's IDs are immediately precede the joining node from its successor [3]. Dependency of our proposal is on node arrival and departure operations to manage file chunks among the nodes.

Lookup latency delays can be reduced by performing discovery of file chunks in parallel. To further reduce latency delay we can use of DHTs that offers one-hop lookup delay (Amazon's dynamo). We can integrate our proposal to existing large scale distributed file systems as well. Our proposal works perfect with both uniform and non-uniform distribution of nodes & file chunks [10] [11].

3.2 Load Rebalacning Algorithm:

A given system is said to be in load balanced state if each chunk server hosts not more than A chunks. Here, each chunk server 1st estimates whether it is under loaded or overloaded without global knowledge. If a given node i departs and rejoins as successor of another node j, then we represent node I as j+1, node j' s original successor as node j+2, the successor of node j' s original successor as node j+3, and so on. If any node in the system is light node then it search for heavy node and takes over at most A chunks from the heavy node.

Table 1 : The Symbols Used In Algorithm

Symbol	Description
| · |	set cardinality
||·||	absolute value function
V	set of chunkservers (storage nodes)
n	|V|
m	number of file chunks stored in V
O	set of heavy (overloaded) nodes
U	set of light (underloaded) nodes
A	ideal number of file chunks hosted by a node
$\bar{A}^i$	estimation of A by node i
L^i	load (number of file chunks) stored in node i ∈ V
V”	vector containing randomly selected nodes
nv	number of vectors collected and maintained by a node
s	|V|
Δ^L and	parameters identifying light and heavy nodes
γ_i	γ approximated by node i

Algorithm 1: SEEK(V,ΔL,ΔU): a light node i seeks
An overloaded node j

Input: vector V = {s samples}, Δ_L and Δ_U
Output: an overloaded node, j

1. $\bar{A}_i$ ← an **1** estimate for A based on { $\bar{A}_j$: j ∈ V};
2. if $L_i < (1 - \Delta_L)\bar{A}_i$ then
3. V ←V ∪{i};
4. sort V according to L_j (∀ $_j$ ∈ V) in ascending order;
5. k ← i's position in the ordered set V;
6. find a smallest subset P ⊂ V such that

(i) $L_j > (1 + \Delta_U)\bar{A}_j$, ∀ $_j$ ∈ P, and

(ii) $\sum_{j\in P}(L_j - \bar{A}_j) \geq k\bar{A}_i$

7. j ← the least loaded node in P;

return j;

Algorithm 2: MIGRATE(i, j): a light node i requests chunks from an overloaded node j
Input: a light node i and an overloaded node j

1. **if** $L_j > (1 + \Delta_U)$ _ **1** $\bar{A}_j$ and j is willing to share its load with i **then**
2. i migrates its locally hosted chunks to i + 1;
3. i leaves the system;
4. i rejoins the system as j's successor by having
5. i ← j + 1;
6. t ← $\bar{A}_i$;
7. **if** t >($L_j - (1 + \Delta_U)\bar{A}_i$) **then**
   t ← $L_j - (1 + \Delta_U)$ _ $\bar{A}_i$;
8. i allocates t chunks with consecutive IDs from j;
9. j removes the chunks allocated to i and renames its ID

In response to the remaining chunks it manages;

Algorithm 3: MIGRATELOCALITYAWARE (i, V”): a light
Node i joins as a successor of a heavy node j that is physically closest to i
Input: a light node i and V = {V1, V2, . . . , V nv }

1. C ← ∅;
2. **for** k = 1 **to** nv **do**
3. C ← C ∪ SEEK(V_k);
4. j ← the node in C physically closest to i;
5. MIGRATE(i, j);

Algorithm 4: SEEKFORHETEROGENEITY (V, Δ_L and Δ_U): a light node i seeks an overloaded node j in an heterogeneous environment where

nodes have different capacities (here, γ_i denotes γ approximated by node i)

In our proposal 1st will propose algorithm in which nodes will have global knowledge of the system & then in 2nd algorithm nodes without global knowledge of the system. With the help of global knowledge if node find that it is light node then node leaves the system by migrating its locally hosted chunks to its successor i+1 & then rejoins quickly to as the successor of heavy node. For relieving the load of heavy node, light node send requests as min {Lj - A, A} chunks from heavy node. That is light node request the exceeded load from heavy node. After this if heavy node still remains as heavy node then another light node performs the same procedure. This process repeats until the heaviest node doesn't remains as heavy node. This process is carried for all heavy nodes and light nodes for load balancing.

This algorithm in this way helps to achieve the load balanced state as quick as possible and also helps to reduce the movement cost since only the light nodes in the system migrates towards heavy node.

We can reduce the time complexity of this algorithm, if every light node knows to which heavy node it needs to request beforehand and then parallel load balancing can be done. For this 1st we sort out the top light nodes and top heavy nodes in the system, mapping of each light node to heavy node is done. In this way all light nodes can concurrently request chunks from heavy node which will help to reduce the latency of sequential algorithm to achieve the load-balanced state.

It is not possible to have global knowledge system in a very large-scale distributed file system so we propose algorithm which will work in distributed manner without global knowledge. Then we try to improve our proposal by taking advantage of physical network locality to reduce network traffic. We have to also consider the nodes with heterogeneous capacity & also the high file availability is asked from large-scale and dynamic distributed storage systems where chances of failure are more. To tackle this issue we try to maintain the replica of each file chunk.

As discussed earlier, we try to propose a algorithm where nodes will not have a global knowledge but it's a very challenging task, so in our system we try to solve this problem by creating a group of nodes by randomly selecting nodes. And then each node contacts the number of nodes in group & builds a vector denoted by V. vector consists of entries, & each entry consists of ID, network address & load statues of nodes in that group. Using gossip-based protocol, each node carries out exchange with its neighbors until its vector has entries [24]. Then it calculates average load of each node and consider it as estimation.

Then if node i is light node then it searches for heavy node for requesting chunks. Then node i performs sorting of nodes including itself in its vector and finds its position in sorted list. Node i find out the overloaded nodes such that it exceeds the maximum load limit or equal to maximum load limit. Node i then request the chunks from heavy node.

But here there might be a case where different nodes try to share the load of node j, for this node j offloads its load to randomly selected node. Also it is possible that number of heavy nodes selects same node to share their loads. In this case light node randomly picks up heavy nodes for reallocation.

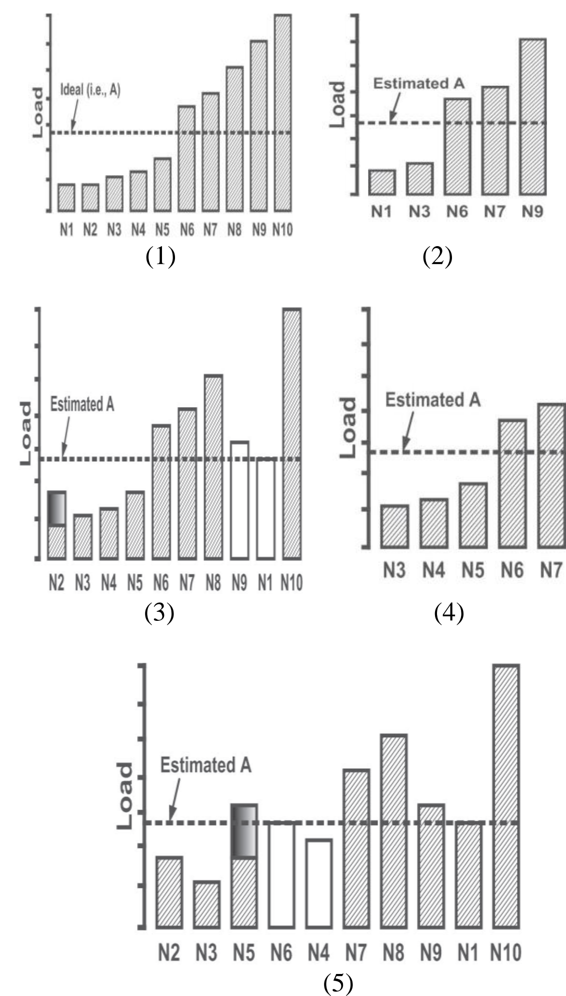


Fig. 2. An example illustrating our algorithm, where (1) the initial loads of chunkservers N1;N2; . . .;N10, Fig.2.2 N1 creates a sample of the loads of N1, N3, N6, N7, and N9 for performing the load rebalancing algorithm, Fig.2.3 N1 leaves and sheds its loads to its successor N2, and then rejoins as N9's successor by allocating AeN1 chunks (the

ideal number of chunks N1 estimates to manage) from N9, Fig.2.4 N4 collects its sample set fN3;N4;N5;N6;N7g, and Fig.2.5 N4 departs and shifts its load to N5, then allocates exceeded chunks from N6 ad rejoins as successor to N6.

3.3 Use Of Physical Network Locality:

DHT network can be useful to find out logical proximity. But it's not useful to find out physical location using logical proximity in practical. That is a message travelling between two neighbors in DHT network may travel a long physical distance through various physical network links. So in this case heavy migration will require for sending message which will induce heavy network traffic and consumption of network resources.

To overcome this problem, instead of creating one vector per algorithmic round, each light node creates NV vectors using the same procedure. Then from NV vectors, node i search NV heavy nodes and then selects physically close heavy node based on message round-trip delay.

For algorithm 3 demonstration consider the above example. Let NV=2. Create two sample sets v1={N1,N3,N6,N7,N9} and v2={N1,N4,N5,N6,N8}. N1 identifies Node N9 and Node N8 in v1 and v2 respectively. Suppose N8 is closer than N9 then node i will join as a successor of N8 & also node i will offload its load to its successor. Further to reduce network traffic we can initialize DHT network such that every two nodes with adjacent

IDs are geometrically close. For this we use space filling curve, which visits each IP address and assigns a unique ID to each address such that geometrically close IP addresses are assigned with numerically close IDs. For invoking the IDs, we can use IP address as input to space filling curve.

3.4 Use Of Node Heterogeneity:

Nodes in the system may be having different capacities in terms of number of file chunks it can accommodate. Consider the capacities of nodes as (C1, C2, C3……, n). We modify our basic algorithm in as each node i approximates the ideal hosting of file chunks in load balanced state as follows:

$\tilde{A}i = \gamma Ci,$

Load per unit capacity is γ which a node should manage in load balanced state and which a node should manage in load balanced state and

$$\gamma = \frac{m}{\sum_{k=1}^{n} Ck}$$

m is number of file chunks.

As we know load of node is directionally proportional to the number of file chunks the node has stored [8]. So we have taken into consideration this while designing. To find out average load per unit capacity we will use gossip-based aggregation protocol [20] [21]. Our basic algorithm is then modified taking into consideration of node heterogeneity.

IV. IMPLEMENTATION

For implementation of our algorithm we will use computer simulations. For this we will use chord and gossip-based aggregation protocols [20]. The number of nodes in the system will be n=1,000 and number of file chunks m=10000. Number of chunks hosted by a node will make use of geometric distribution. Figures will elaborate this concept. We have use standalone load balancing server which will acquire global knowledge of the file chunks in the system from the namenode. These namenodes manages the metadata of complete file system. Now by making use of this global knowledge, it partitions the nodes as overloaded nodes and light nodes, then balancer will randomly pick up the light node and heavy node and balances their nodes according to our algorithm. The reallocation will end when there is no pair of light and heavy node found by balancer. As we have modified our algorithm to reduce network traffic, load balancer will try to reallocate loads of nodes in same rack 1st and if no node in same rack is found by balancer then it will access other new rack for reallocation.

4.1 Implementation Result

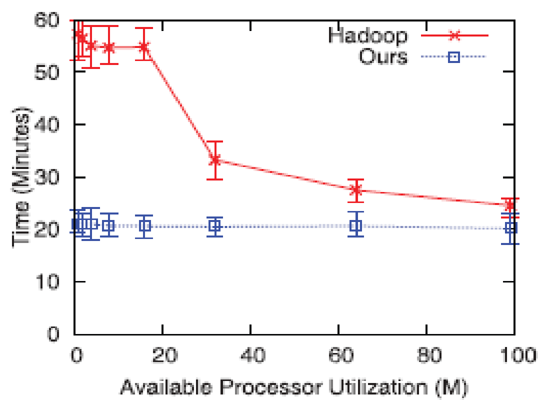
Implementation results will prove that our approach is performs well than centralized approach, since load balancer collects the information from namenode. This is because each node randomly selects other node without global knowledge of the system. Our proposal is distributed and it is not require gaining global knowledge of the system.

Movement cost of our proposal will be very low since we will make use physical network locality. And making use of this information and arrangement of nodes accordingly will help to reduce migration of nodes and file chunks. Also in our proposal only light nodes will offload its load to successor to achieve load balanced state will be low. Because the movement cost require for reallocation of light nodes will be less.

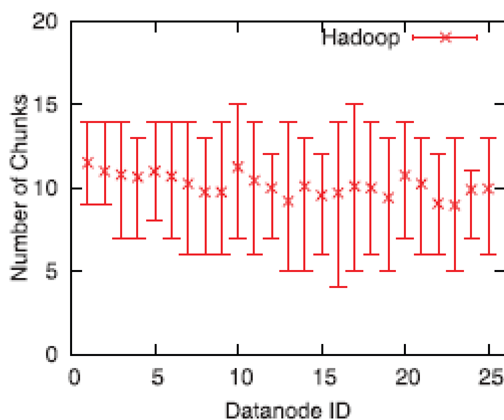
Messages generated by our algorithm are within the limits and are less as compared to other approaches. This will result in less message overload[24]. In our proposal we are depending on a chord protocol. Number of operations required for rejoining and departure in our algorithm are

considerably more than centralized approach but less than other approaches.

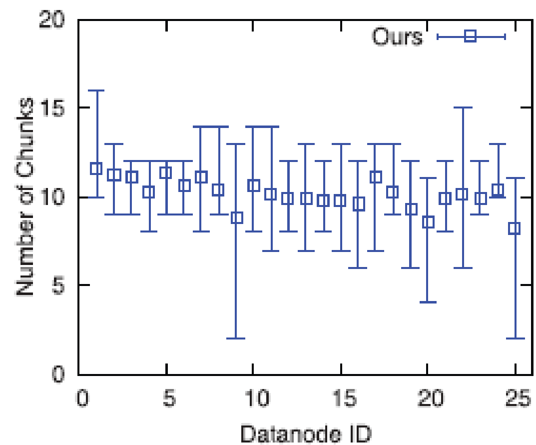
As we have modified our algorithm for making use of physical network locality, we have found that our algorithm will work well in load rebalancing task. Since in our algorithm we will create groups of nodes with physically close to each other and while reallocation we will make use of this information, load balancer will 1st reallocate the nodes from the same rack then if still require then only access the other rack for load balancing [27]. As mentioned by making use of chord ring all adjacent nodes in the ring are physically close also. This helps in achieving fast convergence of system to uniform load balanced state with low network traffic and less consumption of network resources.
Following figure will elaborate the concept of how our algorithm will work



Shows the time elapsed of HDFS load balancer and our proposal



1) Distribution of chunks for HDFS



2) Expected distribution of chunks for our proposal

V. EXPERIMENTAL SETUP

For implementation we are using HDFS 0.21.0 & for assessment of implementation we are using load balancer in HDFS.
For demonstration we will use small cluster environment, which consists of a single dedicated name node and 25 datanodes,
Software requirement: Ubuntu 10.10

Hardware requirement: Intel core 2 duo E7400 processor, 3gb RAM (RAM size depends upon number of file chunks to be processed)

For implementation, a number of clients are established and these clients will issue request to the namenode. Requests like create & delete directories, in our proposal we will set up 6 clients for generating requests. Further we will limit the processor cycles available for namenode by periodically varying maximum processor utilization. When there will be lower processor availability then less number of cycles will be available for namenode to allocate to handle the clients requests.

We will use maximum 256 chunks scattered in the file system for connecting all nodes with 100mbps network. For each execution of algorithm we will calculate the time required to complete load balancing, also for load balancer in HDFS and our proposal. We will perform 20 runs for a given processor utilization and calculate the average time required for algorithm execution. Random sampling of 10 nodes will be carried out.

VI. EXPECTED OUTPUT

Uniform load distribution among all nodes in cloud.

Reduced movement cost.

Network traffic is reduced.

VII. CONCLUSION AND SUMMARY

Our proposed algorithm will prove as efficient approach to tackle load rebalancing problem for large-scale, dynamic distributed file systems in clouds. Our proposal helps to achieve the load balanced state and also reduces the movement cost at far extend, by making a perfect use of node heterogeneity and physical network distribution. We will compare our proposal with existing systems for better assessment of our proposal, for this we will deal with heavy loaded nodes. Our proposal has number of enhancements than the centralized approach or typical distributed approach which will result in high performance load balancing technique.

For future work we can consider issues which require in depth knowledge of issues like metadata management, file consistency models and replication strategies.

REFERENCES

[1]. Chung, H. Yi, Shen, Haiying, Chao, Y. Chang, "Load rebalancing for distributed file system in clouds" , IEEE Trans, Software Engineering., volume 24 no. 5, May 2013.

[2]. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," Proc. 19th ACM Symp. O. S. Principles, pp. 29-43, October 2003.

[3]. K. McKusick and S. Quinlan, "GFS: Evolution on Fast-Forward," Communication. ACM, volume 53, no. 3, pp. 42-49, January 2010.

[4]. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Proc. Sixth Symp. O. S. Design and Implementation, pp. 137-150, December 2004.

[5]. VMware, http://www.vmware.com/, 2012.

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

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

[8]. I.Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," IEEE/ACM Transactions. Networking, volume 11, no. 1, pp. 17-21, February 2003.

[9]. 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, November 2001.

[10]. 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, pp. 205-220, October 2007.

[11]. 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, pp. 68-79, February 2003.

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

[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, pp. 444-455, September 2004.

[14]. J.W. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," Proc. First Int'l Workshop Peer-to-Peer Systems, pp. 80-87, February 2003.

[15]. Y. Zhu and Y. Hu, "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems," IEEE Trans. Parallel and Distributed Systems, volume 16, no. 4, pp. 349-361, April 2005.

[16]. H. Shen and C.-Z. Xu, "Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured P2P Networks," IEEE Trans. Parallel and Distributed Systems, volume 18, no. 6, pp. 849-862, June 2007.

[17]. 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, volume 22, no. 4, pp. 634-649, April 2011.

[18]. M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.

[19]. D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, September 2001.

[20]. M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-Based Aggregation in Large Dynamic Networks," ACM Trans. Computer Systems, volume 23, no. 3, pp. 219-252, August 2005.

[21]. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M.V. Steen, "Gossip-Based Peer Sampling," ACM Trans. Computer Systems, volume 25, no. 3, August. 2007.

[22]. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang , S. Lu, "BCube: A High Performance, Server-Centric Network Architecture for Modular Data Centers," Proc. ACM SIGCOMM '09, pp. 63-74, August 2009.

[23]. H. Abu-Libdeh, P. Costa, A. Rowstorn, G. O.Shea, and A. Donnelly, "Symbiotic Routing in Future Data Centers," Proc. ACM SIGCOMM 10, pp. 51-62, August 2010.

[24]. S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," Performance Evaluation, volume 63, pp. 217-240, March 2006.