International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 10, Issue 11 (November 2014), PP.51-57

Load Rebalancing with Security for Hadoop File System in Cloud

[1]Vidya N. Chiwande, Prof. Animesh R. Tayal[2]
Computer Technology Priyadarshini College of Engineering Nagpur, India

Abstract:- [1]A file system is used for the organization, storage,[1]retrieval, naming, sharing, and protection of files. Distributed file system has certain degrees of transparency to the user and the system such as access transparency,[2] location transparency, failure transparency, heterogeneity, replication transparency etc. [1][3]NFS (Network File System), RFS (Remote File Sharing), Andrew File System (AFS) are examples of Distributed file system. Distributed file systems are generally used for cloud computing applications based on [4] the MapReduce programming model. A MapReduce program consist of a Map () procedure that performs filtering and a Reduce () procedure that performs a summary operation. However, in a cloud computing environment, sometimes failure is occurs and nodes may be upgraded, replaced, and added in the system. Therefore load imbalanced problem arises. To solve this problem, load rebalancing algorithm is implemented in this paper so that central node should not overloaded. The implementation is done in hadoop distributed file system. As apache hadoop is used, security issues are arises. To solve these security issues and to increase security, [20] Kerberos authentication protocol is implemented to handle multiple nodes. This paper shows real time implementation experiment on cluster with result.

Keywords:- Hadoop , Distributed File System, Load Rebalance , Kerberos, Cloud.

I. INTRODUCTION

The[1]cloud computing refers to the development and implementation of models for enabling omnipresent, convenient, on-demand access to a shared set of configurable computing resources (e.g. networks, servers, storage, applications, and services). This encompasses the consideration of network access techniques that guarantee fluid service provider interaction with the cloud users [2]. In the associated business model, users only pay only for the services they actually use, without prior commitment, enabling cost reductions in IT deployment and a scalability of far greater resources, which are abstracted to users in order to appear unlimited, and presented through a simple interface that hides the back-office processes[1]. There are three cloud-based service models have been proposed [2].

i) *Software as a Service (SaaS)**:*** providing applications running in the cloud, where the customer has virtually no access control or management of the internal infrastructure.

ii) *Platform as a Service (PaaS)*: providing a set of tools that support certain technologies of development and the entire necessary environment for deploying applications created by the customer, who is able to control and manage them.

iii) *Infrastructure as a Service (IaaS)*: providing basic computing resources such as processing, storage and network bandwidth where the client can run any operating.

Cloud computing has improved performance, reduced software cost, instant software updates, improved document format compatibility, unlimited storage capacity etc.

Distributed file system [3] support hybrid mode of cloud. It plays very important role for cloud computing applications which is based on the MapReduce programming model. In such a file system, nodes simultaneously perform computing and Storage function, file is divided into number of chunks allocated in different nodes. A file system is responsible for the organization, storage, retrieval, naming, sharing, and Protection of files. File systems provide directory services, which convert a file name (possibly a hierarchical one) [4] into an internal identifier (e.g. inode, FAT index). They contain a representation of the file data itself and methods for accessing it (read/write). [2]The file system is responsible for controlling access to the data and for performing low-level operations such as buffering frequently used data and issuing disk I/O requests. Distributed file system are to present certain [3] degrees of transparency to the user and the system.

A file service [1] is a specification of what the file system offers to clients. A file server is the implementation of a file service and runs on one or more machines. A file itself contains a name, data, and attributes (such as owner, size, creation time, access rights). An immutable file is one that, once created, cannot be changed. Immutable files are easy to cache and to replicate across servers since their contents are guaranteed to remain unchanged.

[20]MapReduce is A programming model for large-scale distributed data processing which is Simple, elegant ,concept Restricted, yet powerful programming construct Building block for other parallel programming tools Extensible for different applications Also an implementation of a system to execute such programs Take

advantage of parallelism ,Tolerate failures and jitters Hide messy internals from users Provide tuning knobs for different applications. MapReduce is the heart of hadoop, In the MapReduce model, computation is divided into a map function and a reduce function. [5]The map function takes a key/value pair and produces one or more intermediate key/value pairs it means it takes a set of data and convert it into another set of data where individual element are broken down into tuples (key/value pairs). The reduce function then takes these intermediate key/value pairs and merges all values corresponding to a single key means The reduce job takes output from map as input and combine those data tuples into smaller set of tuples.

Apache Hadoop[2] is a distributed system for storing large amounts of data and processing the data in parallel. Hadoop is used as a multi-tenant service internally at Yahoo! and stores sensitive data such as personally identifiable information or financial data. Other organizations, including financial organizations, using Hadoop are beginning to store sensitive data on Hadoop clusters. [1]As a result, strong authentication and authorization is necessary. Hadoop has been under development at Yahoo! And a few other organization as an Apache open source project over the last 5 years. It is gaining wide use in the industry. Yahoo!, for example, has deployed tens of Hadoop clusters, each typically with 4,000 nodes and 15 petabytes. Hadoop contains two main components. The first component, is a distributed file system similar to GFS. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

HDFS has a master/slave architecture.[11] A HDFS cluster consists of a single NameNode and a number of DataNodes. The NameNode is a master server that manages the file system namespace and regulates access to files by clients. The DataNodes manage storage attached to the nodes that they run on. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and [13]write requests from the file system's clients.

II. LOAD REBALANCING PROBLEM

Consider a large-scale distributed file system[5] consisting of a set of chunkservers V in a cloud, where the cardinality of V is |V| = n. typically, n can be 1,000, 10,000, or more. In the system, a number of files are stored in the n chunkservers. First, let us denote the set of files as F. Each file f∈F is partitioned into a number of disjointed, fixed size chunks denoted by C_f . For example, each chunk has the same size, 64 Mbytes, in Hadoop HDFS [3]. Second, the load of a chunkserver is proportional to the number of chunks hosted by the server [3]. Third, node failure is the norm in such a distributed system, and the chunkservers may be upgraded, replaced and added in the system. Finally, the files in F may be arbitrarily created, deleted, and appended. The net effect results in file chunks not being uniformly distributed to the chunkservers. Fig. illustrates an example of the load rebalancing problem with the assumption that the chunkservers are homogeneous and have the same capacity. Our objective in the current study is to design a load rebalancing algorithm to reallocate file chunks such that the chunks can be distributed to the system as uniformly as possible while reducing the movement cost as much as possible. Here, the movement cost is defined as the number of chunks migrated to balance the loads of the chunkservers. Let A be the ideal number of chunks that any chunkserver i∈V is required to manage in a system-wide load-balanced state, that is,

$$A=\frac{\sum_{f\epsilon F}|C|}{n} \quad (1)$$

Then, load rebalancing algorithm aims to minimize the load imbalance factor chunkserver as follows:

$$\|L_i\text{-}A\| \quad (2)$$

where Li denotes the load of node i (i.e. the number of file chunks hosted by i) and ‖·‖ represents the absolute value function.

III. OUR PROPOSAL

1. Load Balancing Algorithm

1.1. Overview

*[1]*A large-scale distributed file system is in a load-balanced state if each chunkserver hosts no more than A chunks. In our proposed algorithm, each chunkserver node i first estimate whether it is underloaded (light) or overloaded (heavy) without global knowledge. A node is light if the number of chunks it hosts is smaller than the threshold of $(1-\Delta_L)A$ (where $0\leq \Delta_L < 1$). In contrast, a heavy node manages the number of chunks greater than $(1+\Delta_U)A$, where $0 \leq \Delta_U < 1$. Δ_L and Δ_U are system parameters. In the following discussion, if a node i departs and rejoins as a successor of another node j, then we represent node i as node 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. For each

node i∈V , if node i is light, then it seeks a heavy node and takes over at most A chunks from the heavy node. Here first present a load-balancing algorithm, in which each node has global knowledge regarding the system, that leads to low movement cost and fast convergence. We then extend this algorithm for the situation that the global knowledge is not available to each node without degrading its performance. [1]Based on the global knowledge, if node i finds it is the least-loaded node in the system, i leaves the system by migrating its locally hosted chunks to its successor i + 1 and then rejoins instantly as the successor of the heaviest node (say, node j). To immediately relieve node j's load, node i requests min {Lj–A, A} chunks from j. That is, node i requests A chunks from the heaviest node j if j's load exceeds 2A; otherwise, i requests a load of Lj -A from j to relieve j's load.

Node j may still remain as the heaviest node in the system after it has migrated its load to node i. In this case, the current least-loaded node, say node i' departs and then rejoins the system as j's successor. That is, 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.

The mapping between the lightest and heaviest nodes at each time in a sequence can be further improved to reach the global load-balanced system state. [2]The time complexity of the above algorithm can be reduced if each light node can know which heavy node it needs to request chunks beforehand, and then all light nodes can balance their loads in parallel. Thus, we extend the algorithm by pairing the top-k_1 underloaded nodes with the top-k_2 overloaded nodes. We use U to denote the set of top-k_1 underloaded nodes in the sorted list of underloaded nodes, and use O to denote the set of top-k_2 overloaded nodes in the sorted list of overloaded nodes. Based on the above-introduced load balancing algorithm, the light node that should request chunks from the k_2' th ($k_2' \leq k_2$) most loaded node in O is the k_1'th ($k_1' \leq k_1$) least loaded node in U, and

$$k_1' = \frac{\sum_{ith\ most\ \ loaded\ \ node\ \in O}^{k_2'} (L_i - A)}{A} \qquad (3)$$

Where $\frac{\sum_{ith\ most\ \ loaded\ \ node\ \in O}^{k_2'} (L_i - A)}{}$ denotes the sum of the excess loads in the top-k_2' heavy nodes. It means that the top-k_1' light nodes should leave and rejoin as successors of the Top-k_2' overloaded nodes. Thus, according to (3), based on its position k_1' in U, each light node can compute k_2' to identify the heavy node to request chunks. Light nodes concurrently request chunks from heavy nodes, and this significantly reduces the latency of the sequential algorithm in achieving the global system load-balanced state.

2. Basic Load Rebalancing Algorithm

[1]Algorithm 1 specifies the operation that a light node i seeks an overloaded node j, and Algorithm 2 shows that i requests some file chunks from j. Without global knowledge, pairing the top-k1 light nodes with the top-k2 heavy nodes is clearly challenging.. In the basic algorithm, each node implements the gossip-based aggregation protocol to collect 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 denoted by V. A vector consists of entries, and each entry contains the ID, network address and load status of a randomly selected node. Using the gossip-based protocol, each node i exchanges its locally maintained vector with its neighbors until its vector has s entries. It then calculates the average load of the s nodes denoted by A_i and regards it as an estimation of A (Line 1 in Algorithm 1).

Symbol	Description
|.|	Set cardinality
‖.‖	Absolute value function
V	Set of chunkserver
n	|V|
m	Number of file chunk store in V
o	Set of heavy node
u	Set of light node
A	Ideal number of file chunk hosted by a node
A_i	Estimation of A by node i
L_i	Load(number of chunk)stored in node i∈V
v	Vector containing randomly selected node
n_v	Number of vector collected and maintain by node
8	|V|
ΔL ΔU	Parameter identifying light and heavy nodes

Table 3.1: Symbols to be used

input : vector $\mathcal{V} = \{s \text{ samples}\}$, Δ_L and Δ_U
output : an overloaded node, j

1 $\widetilde{\mathcal{A}}_i \leftarrow$ an estimate for $\mathcal{A}$ based on $\{\widetilde{A}_j : j \in \mathcal{V}\}$;
2 **if** $L_i < (1-\Delta_L)\widetilde{\mathcal{A}}_i$ **then**
3 $\quad \mathcal{V} \leftarrow \mathcal{V} \cup \{i\}$;
4 $\quad$ sort $\mathcal{V}$ according to $L_j (\forall j \in \mathcal{V})$ in ascending order;
5 $\quad k \leftarrow i$'s position in the ordered set $\mathcal{V}$;
6 $\quad$ find a smallest subset $\mathcal{P} \subset \mathcal{V}$ such that
$\quad$ (i) $L_j > (1+\Delta_U)\widetilde{\mathcal{A}}_j, \forall j \in \mathcal{P}$, and
$\quad$ (ii) $\sum_{j \in \mathcal{P}} \left(L_j - \widetilde{\mathcal{A}}_j\right) \geq k\widetilde{\mathcal{A}}_i$;
7 $\quad j \leftarrow$ the least loaded node in $\mathcal{P}$;
8 $\quad$ **return** j;

Algorithm 1: SEEK$(\mathcal{V}, \Delta_L, \Delta_U)$: a light node i seeks an overloaded node j

If node i finds itself is a light node (Line 2 in Algorithm 1), it seeks a heavy node to request chunks. [1]Node i sorts the nodes in its vector including itself based on the load status and finds its position k_1' in the sorted list, i.e., it is the top-k1 underloaded node in the list (Lines 3-5 in Algorithm 1). Node i finds the top-k_2' overloaded nodes in the list such that the sum of these nodes' excess loads is the least greater than or equal to k_1'Ái (Line 6 in Algorithm 1). Formula (ii) in the algorithm is derived from (3). The complexity of the step in Line 6 is O(|V|). Then, the k_2'th overloaded node is the heavy node that node i needs to request chunks (Line 7 in Algorithm 1).[1] Considering the step in Line 4, the overall complexity of Algorithm 1 is then O (|V| log|V|).

input : a light node i and an overloaded node j
1 **if** $L_j > (1+\Delta_U)\widetilde{\mathcal{A}}_j$ and j is willing to share its load with i **then**
2 $\quad i$ migrates its locally hosted chunks to $i+1$;
3 $\quad i$ leaves the system;
4 $\quad i$ rejoins the system as j's successor by having $i \leftarrow j+1$;
5 $\quad t \leftarrow \widetilde{\mathcal{A}}_i$;
6 $\quad$ **if** $t > \left(L_j - (1+\Delta_U)\widetilde{\mathcal{A}}_i\right)$ **then**
7 $\quad\quad t \leftarrow L_j - (1+\Delta_U)\widetilde{\mathcal{A}}_i$;
8 $\quad i$ allocates t chunks with consecutive IDs from j;
9 $\quad j$ removes the chunks allocated to i and renames its ID in response to the remaining chunks it manages;

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

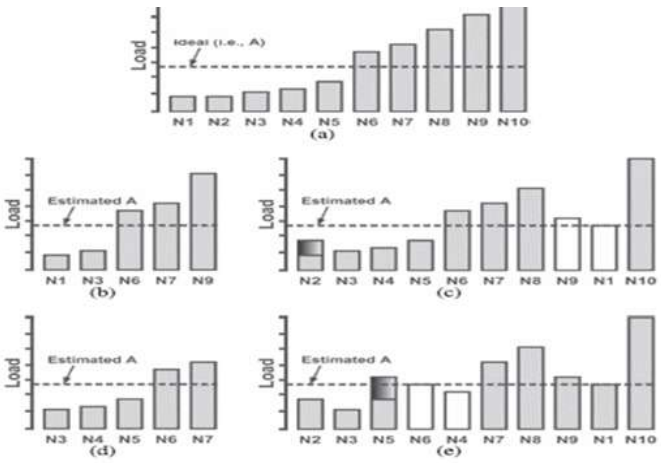
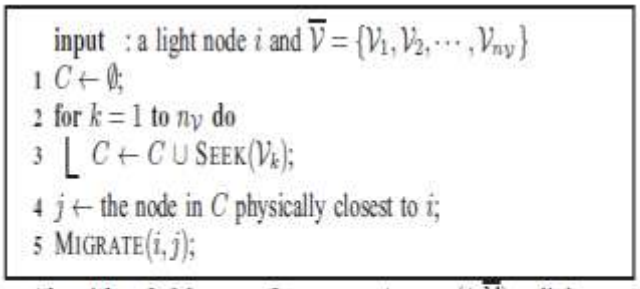


Figure 3: In the example illustrate above, where (a)the initial loads of chunkserver N1,N2…….N10 (b)N1 samples the loads of N1,N3,N6,N7 and N9 in order to perform the load rebalancing algorithm. (c) N1 leaves and sheds its load to its successor N2, and then rejoins as N9's successor by allocating ideal number of chunks from N9,(d) N4 collect its sample set{N3,N4,N5,N6,N7},and (e) N4 departs and shift its load to N5,and then rejoin as the successor of N6 by allocating chunks from N6.

In the example, N4 also performs the load rebalancing algorithm by first sampling {N3;N4;N5;N6;N7} . Similarly, N4 determines to rejoin as the successor of N6. N4 then migrates its load to N5 and rejoins as the successor of N6 .N4 requests min{LN6 -AeN4;AeN4}= L6 -AeN4 chunks from N6.



Algorithm 3: MIGRATELOCALITYAWARE$(i, \overline{V})$: a light node i joins as a successor of a heavy node j that is physically closest to i

In Algorithm 3 ,Lines 2 and 3 take O(n_v |V| log |V|). A larger n_v introduces more overhead for message exchanges, but results in a smaller movement cost.

3. Taking Advantage of Node Heterogeneity

[1][2] Nodes participating in the file system are possibly heterogeneous in terms of the numbers of file chunks that the nodes can accommodate. We assume that there is one bottleneck resource for optimization although a node's capacity in practice should be a function of computational power, network bandwidth, and storage space. Given the capacities of nodes (denoted by $\{\beta_1, \beta_2, \beta_3 \ldots\ldots \beta_n\}$), we enhance the basic algorithm in Section 3.2.2 as follows: each node i approximate the ideal number of file chunks that it needs to host in a load balanced state as follows:

$$\check{A}=\text{ɤ}\beta_i \qquad (4)$$

Where ɤ is the load per unit capacity a node should manage the load balanced state and

$$\text{ɤ}=\frac{m}{\sum_{k=1}^{n}\beta_k} \qquad (5)$$

Where m is the number of file chunks stored in the system. As mentioned previously, in the distributed file system for MapReduce [20]based applications, the load of a node is typically proportional to the number of file chunks the node possesses [3]. Thus, the rationale of this design is to ensure that the number of file chunks managed by node i is Proportional to its capacity. To estimate the aggregate ɤ, our proposal again relies on the gossip-based aggregation protocol in computing the value.

Algorithm 4[1] presents the enhancement for Algorithm 1 to exploit node heterogeneity, which is similar to Algorithm 1 and is self-explanatory. If a node i estimates that it is light (i.e., Li < (1-Δ_L) $\check{A}i$), i then rejoins as a successor of a heavy node j. i seeks j based on its sampled node set V. i sorts the set in accordance with Lt/β_t , the load per capacity unit a node currently receives, for all t∈V. When node i notices that it is the kth least-loaded node (Line 6 in Algorithm 4), it then identifies node j and rejoins as a successor of node j. Node j is the least-loaded node in the set of nodes PCV having the minimum cardinality, where 1) the nodes in P are heavy, and 2) the total excess load of nodes in P is not less than $\sum_{jth\ light\ node\ in\ V}^{k} \check{A}j$ (Line 7 in Algorithm 4). Here, $\sum_{jth\ light\ node\ in\ V}^{k} \check{A}j$ indicates the sum of loads that the top-k light nodes in V will manage in a load balanced system state

input : vector $\mathcal{V} = \{s \text{ samples}\}$, Δ_L and Δ_U
output : an overloaded node, j

$\tilde{\gamma}_i \leftarrow$ an estimate for γ based on $\{\tilde{\gamma}_j : j \in \mathcal{V}\}$;
$\tilde{\mathcal{A}}_i \leftarrow \tilde{\gamma}_i \beta_i$;
if $L_i < (1 - \Delta_L)\tilde{\mathcal{A}}_i$ **then**
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{i\}$;
  sort $\mathcal{V}$ according to $\frac{L_j}{\beta_j} (\forall j \in \mathcal{V})$ in ascending order;
  $k \leftarrow i$'s position in the ordered set $\mathcal{V}$;
  find a smallest subset $\mathcal{P} \subset \mathcal{V}$ such that
  (i) $L_j > (1 + \Delta_U)\tilde{\mathcal{A}}_j, \forall j \in \mathcal{P}$, and
  (ii) $\sum_{j \in \mathcal{P}} \left(L_j - \tilde{\mathcal{A}}_j\right) \geq \sum_{j=1}^{k} \tilde{\mathcal{A}}_j$;
  $j \leftarrow$ the least loaded node in $\mathcal{P}$;
  return j;

Algorithm 4: SEEKFORHETEROGENEITY($\mathcal{V}, \Delta_L, \Delta_U$): a light node i seeks an overloaded node j in an heterogeneous environment where nodes have different capacities (here, $\tilde{\gamma}_i$ denotes γ approximated by node i)

IV. EXPERIMENTAL WORK

A. Kerberos Protocol

[16]Kerberos is an authentication protocol that allows clients and servers to reliably verify each others identity before connectivity can happen. [17]It provides advantages such as mutual authentication and message integrity as well as data confidentiality. Kerberos must go through a process of establishing a secure authenticated network connection. [18]This process is performed as client and server validate their respective identities to each other before performing any application functions. Both the client and the server must establish a "trust" before a network connection can be established. [16]In practical terms this means that the service must be able to determine who the client is without asking the client and the client must be able to determine who the service is without asking the service.

The Process

1. Client ask authentication server for ticket to Ticket Granting Server (TGS)

Authentication server look for the client in the database the n it generate session key(SK1) for use between client and TGS. Kerberos encrypt the SK1 using client's secret key. Authentication server can create and send ticket granting ticket (TGT) to client by using TGS's secret key.

2. Ticket Granting Service Exchange

The client decrypt the message and recover the session key then it uses to create authenticator containing the user name ,IP address and timestamp.client then send this authenticator along with TGT to he TGS for requesting access to the target server.the TGS decrypt the TGTthen uses the SK1 inside the TGT to decrypt authenticator. [17]It verify the information inside the authenticator if evrything is matches then it proceed the request.after that TGS create new session key SK2 for client and target server, encrypt it using SK1 and send it to client.

3. Client/Server exchange

[17]The client decrypt the message and gets SK2.the client create new authenticator encrypted with SK2.the client send session ticket and authenticator.the target server decrypt and check the ticket ,authenticator,client address and timestamp.

4. Secure Communication

[18]The target server knows the client and share encryption key for secure communication because only client and target server share this key.

B. Result

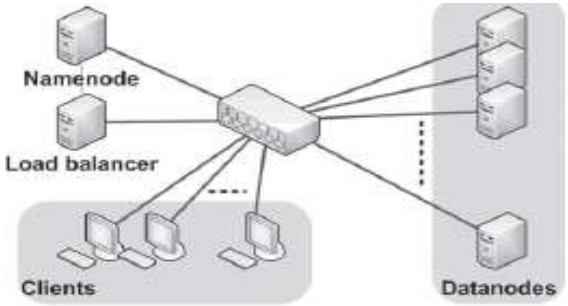
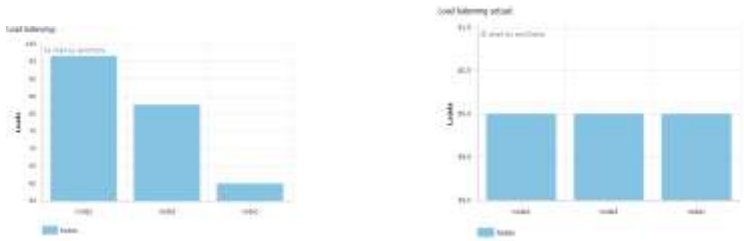


Figure 4.1: Experimental set up of cluster



Graph 4.1(a): load before balancing (b): Load after balancing

ACKNOWLEDGMENT

The authors are grateful to reviewers who have provided valuable comments to improve study.

Ms.Vidya N. Chiwande , she is student of Computer Technology dept. at PCE Nagpur India.
Prof. Animesh R. Tayal , he is Assistant Professor in Computer Technology dept of PCE Nagpur India.

REFERENCES

[1]. Hung-Chang Hsiao, Member, IEEE Computer Society, 18Hsueh-Yi Chung, Haiying Shen, Member, IEEE, and Yu-Chang Chao "Load Rebalancing for Distributed File Systems in Clouds", may 2013.
[2]. 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, June2012
[3]. 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
[4]. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI, pages 137–150, 2004.
[5]. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System" , In Proceedings of the nineteenth ACM symposium on Operating systems principles, SOSP '03, pages 29–43, New York, NY, USA, 2003
[6]. Raicu, I.T. Foster, and P. Beckman, "Making a Case for Distributed File Systems at Exascale," Proc. Third Int'l Workshop Large-Scale System and Application Performance (LSAP '11), pp. 11-18, June 2011
[7]. H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic Routing in Future Data Centers," Proc. ACM SIGCOMM'10, pp. 51-62, Aug. 2010.
[8]. K. McKusick and S. Quinlan, "GFS: Evolution on Fast-Forward,"Comm. ACM, vol. 53, no. 3, pp. 42-49, Jan. 2010
[9]. 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
[10]. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C Tian, Y. Zhang,and S. Lu, "BCube: A High performance, Server-Centric Network Architecture For Modular Data Centers,"Proc.ACM SIGCOMM'09, pp 63-74, Aug. 2009
[11]. 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.
[12]. 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
[13]. 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
[14]. 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),July 2004.
[15]. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," Proc. ACM IGCOMM '04, Aug. 2004
[16]. Devaraj Das, Owen O'Malley, Sanjay Radia, and Kan Zhang "Adding Security to Apache Hadoop" Hortonworks, IBM.
[17]. Sufyan T. Faraj Al-Janabi and Mayada Abdul-salam Rasheed "Public-Key Cryptography Enabled Kerberos Authentication" 2011 Developments in E-systems engineering.
[18]. Jose L. Marquez "Kerberos-secure authentication " ,volume 1.2D
[19]. Owen O'Malley, Kan Zhang, Sanjay Radia, Ram Marti, and Christopher Harrell "Hadoop Security Design".
[20]. Jeffrey Dean and Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Clusters".Tom White foreword by Doug Cutting "Hadoop: The Definitive Guide".