ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 4, Issue 5, May 2015*

# Secured Load Rebalancing In Cloud

**Bhavya.G[1], Chaitra.B.N[2], Chaitra.G[3], Shruthi.C[4]**

Assistant Professor, Dept of Information Science, BMSIT, Bangalore, India[1]

B.E Student, Dept of Information Science, BMSIT, Bangalore, India[2,3,4]

**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. 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 proposal, a fully distributed load rebalancing algorithm is presented to cope with the load imbalance problem.
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. In the proposed system we also provide security for the data stored on cloud through encryption and decryption concepts.

**Keywords*:*** cloud computing, distributed file systems, map reduce, Load imbalance

## I. 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 [2], distributed file systems [3], [4] virtualization [5], [6], 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 into a number of chunks allocated in distinct nodes so that Map Reduce tasks can be performed in parallel over the nodes. For example, consider a word count application that counts the number of distinct words and the frequency of each unique word in a large file. 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 [4]. 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 [8], 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 [3] and Hadoop HDFS [4]) 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 (e.g.,[9]) 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. Thus, depending on the central nodes to tackle the load imbalance problem exacerbate their heavy loads. Even with the latest development in distributed file systems, the central nodes may still be overloaded. For example, HDFS federation [10] suggests architecture with multiple name nodes (i.e., the nodes managing the metadata information). Its file system namespace is statically and manually partitioned to a number of name nodes. However, as the workload experienced by the name nodes may change over time and no adaptive workload consolidation and/or migration scheme is offered to balance the loads among the name nodes, any of the name nodes may become the performance bottleneck.

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


DOI 10.17148/IJARCCE.2015.4553

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 4, Issue 5, May 2015*

newly added to sustain the overall system performance [3], [4], 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. The storage nodes are structured as a network based on distributed hash tables (DHTs), e.g.[11][12][13], discovering a file chunk can simply refer to rapid key lookup in DHTs, given that a unique handle (or identifier) is assigned to each file chunk. DHTs enable nodes to self-organize and -repair while constantly offering lookup functionality in node dynamism, simplifying the system provision and management.

In summary, our contributions are threefold as follows: By leveraging DHTs, we present a load rebalancing algorithm for distributing file chunks as uniformly as possible and minimizing the movement cost as much as possible. Particularly, our proposed algorithm operates in a distributed manner in which nodes perform their load-balancing tasks independently without synchronization or global knowledge regarding the system.

Load-balancing algorithms based on DHTs have been extensively studied. However, most existing solutions are designed without considering both movement cost and node heterogeneity and may introduce significant maintenance network traffic to the DHTs. In contrast, our proposal not only takes advantage of physical network locality in the reallocation of file chunks to reduce the movement cost but also exploits capable nodes to improve the overall system performance. Additionally, our algorithm reduces algorithmic overhead introduced to the DHTs as much as possible.

In our proposal we are also implementing security measures for the data stored on the cloud through encryption. We are using AES encryption algorithm.

Our proposal is assessed through computer simulations. The simulation results indicate that although each node performs our load rebalancing algorithm.

Independently without acquiring global knowledge, our proposal is comparable with the centralized approach in Hadoop HDFS and remarkably outperforms the competing distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead. Additionally, our load-balancing algorithm exhibits a fast convergence rate. We derive analytical models to validate the efficiency and effectiveness of our design. Moreover, we have implemented our load-balancing algorithm in HDFS and investigated its performance in a cluster environment.

## II.LITERATURE SURVEY

*A. A fast adaptive load balancing method:*

This method proposed a binary tree structure that is used to partition the simulation region into sub-domains. The characteristic of this fast adaptive balancing method is to adjust the workload between the processors from local areas to global areas. According to the difference of workload, the arrangements of the cells are obtained. But the main workload concentrates on certain cells so that the procedure of adjusting the vertices of the grid can be very long. This problem can be avoided by the fast load balancing adaptive method. Here the region should be partitioned by using the binary tree mode, so that it contains leaf nodes, child nodes, parent nodes etc. There were partition line between the binary tree and the indexes of the cells on the left are smaller than that of right and the indexes on the top are smaller than the bottom. Calculate the workload based on the balancing algorithm. This algorithm has a faster balancing speed, less elapsed time and less communication time cost of the simulation procedure [17].

Advantages of this method are:

- Relative smaller communication overhead relative smaller communication overhead,
- Faster balancing speed, and
- High efficiency.

The main disadvantage is:

- It cannot maintain the topology that is neighboring cells cannot be maintained.

*B. Honey Bee Behaviour Inspired Load Balancing:*

This method proposed an algorithm named honeybee behaviour inspired load balancing algorithm. Here in this method load balancing is done across the virtual machines for maximizing the throughput. The load balancing in cloud computing can be achieved by modelling the foraging behaviour of honey bees. This algorithm is derived from the behaviour of honey bees that uses the method to find and reap food. In bee hives, there is a class of bees called the scout bees and the another type was forager bees .The scout bee which forage for food sources, when they find the food, they come back to the beehive to advertise this news by using a dance called waggle/tremble/vibration dance. The purpose of this dance, gives the idea of the quality and/or quantity of food and also its distance from the beehive. Forager bees then follow the Scout Bees to the location that they found food and then begin to reap it. After that they return to the beehive and do a tremble or vibration dance to other bees in the hive giving an idea of how much food is left. The tasks removed from the overloaded Virtual machines (VMs) act as Honey Bees. Upon submission to the under load VM, it will update the number of various priority tasks and load of tasks assigned to that VM. This information will be helpful for other tasks , i.e., whenever a high priority has to be submitted to VMs, it should consider the VM that has a minimum number of high priority tasks so that the particular task will be executed earlier. Since all VMs are sorted in an ascending order, the task removed will be submitted to under loaded VMs. Current workload of all available VMs can be calculated based on the information received from the data centre [17].

Advantages are:

- maximizing the throughput,
- waiting time on task is minimum and
- Overhead become minimum.

The disadvantage is:

- If more priority based queues are there then the lower priority load will stay continuously in the queue.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 4, Issue 5, May 2015*

*C. Heat Diffusion Based Dynamic Load Balancing:*

This method proposed an efficient cell selection scheme and two heat diffusion based algorithm called global and local diffusion. In distributed virtual environments, various numbers of users and the load accessing by the concurrent users cause a problem. This can be avoided by this algorithm. According to the heat diffusion algorithm, the virtual environment is divided into large number of square cells and each square cell having objects. The working of the heat diffusion algorithm is in such a way that every node in the cell sends load to its neighboring nodes in every iteration and the transfer was the difference between the current nodes to that of neighboring node. So it was related to heat diffusion process. That is the transfer of heat from high to low object, when they were placed adjacently. In local diffusion algorithm, there were local decision making and efficient cell selection schemes are used. Here they simply compared the neighboring node loads to the adjacent node loads. If load is small then the transfer of load becomes possible. When global diffusion algorithm considered, it has two stages that is global scheduling stage and local load migration stage. From various experimental results the global diffusion algorithm becomes the better one [17].

Advantages are:

- Communication overhead is less,
- high speed and
- Require little amount of calculations.

Disadvantages are:

- network delay is high and
- Several iterations are taken so there was a waste of time.

*D. Load Balancing in Dynamic Structured P2P Systems:*

This method proposed an algorithm for load balancing in dynamic peer-to-peer system and other hybrid environments. In most peer-to-peer system the non uniform of objects in the space and also the load of the node can be changed continuously due to the insertion, deletion and other various operations. This leads to decrease the performance of the system. So the concept of virtual server can be introduced. In this proposed load balancing algorithm, the load information of the peer nodes is stored in different directories. These directories help to schedule reassignment of the virtual servers' to develop a better balance. Greedy heuristic algorithm used to find out a better solution for the proper utilization of the nodes. The huge number of virtual servers in the system helps to increase the utilization. The various load information in to the corresponding pool and then the virtual server assignments are to be done. This proposed algorithm should be applied to different types of resources like storage, bandwidth etc, It was designed to handle the various situations like varying load of the node, node capacity, entering and leaving of nodes and also insertion and deletion of the nodes [17].

Advantages are:

- high node utilization and
- Increasing scalability.

Disadvantage is:

- The reassignment of the virtual server is difficult.

## III.LOAD REBALANCING PROBLEM

We consider a large-scale distributed file system consisting of a set of chunk servers 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 chunk servers. First, let us denote the set of files as F. Each file f 2 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 [4]. Second, the load of a chunk server is proportional to the number of chunks hosted by the server [4]. Third, node failure is the norm in such a distributed system, and the chunk servers 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 chunk servers. Fig. 1 illustrates an example of the load rebalancing problem with the assumption that the chunk servers 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 chunk servers.

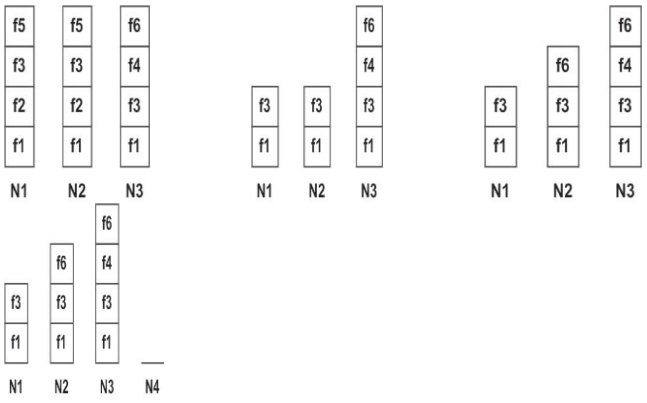



Fig. 1 [1]. An example illustrates the load rebalancing problem, where (a) an initial distribution of chunks of six files $f_1$, $f_2$, $f_3$, $f_4$, $f_5$, and $f_6$ in three nodes $N_1$, $N_2$, and $N_3$, (b) files $f_2$ and $f_5$ are deleted, (c) $f_6$ is appended, and (d) node $N_4$ joins. The nodes in (b), (c), and (d) are in a load-imbalanced state.

## III.LOAD REBALANCING ALGORITHM

In the algorithm, each node implements the *gossip-based aggregation protocol*[14] in 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. Each chunk server nodefirst estimates whether it is under loaded (light) or overloaded (heavy) without global knowledge. Based on the global knowledge, if node i finds it is the least-loaded node in the system, i leaves the system

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 4, Issue 5, May 2015*

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-T; T] chunks from j. 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.

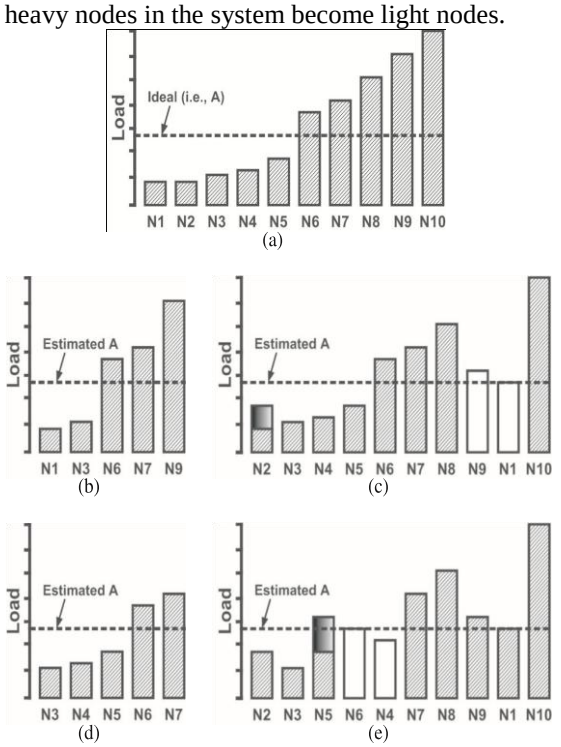



Fig 2 [1]. An example illustrating our algorithm, where (a) the initial loads of chunk servers N1;N2; . . .;N10, (b) N1 samples the loads of N1, N3, N6, N7, and N9 in order to perform the load rebalancing algorithm, (c) N1 leaves and sheds its loads to its successor N2, and then rejoins as N9's successor by allocating AeN1 chunks (the ideal number of chunks N1 estimates to manage) from N9, (d) N4 collects its sample set {N3;N4;N5;N6;N7}, and (e) N4 departs and shifts its load to N5, and it then rejoins as the successor of N6 by allocating L6 _ AeN4 chunks from N6.

Fig. 2 depicts a working example of our proposed algorithm. There are n=10 chunk servers in the system; the initial loads of the nodes are shown in Fig. 2a. Assume $\Delta_l=\Delta_v=0$ in the example. Then, nodes N1, N2, N3, N4, and N5 are light, and nodes N6, N7, N8, N9, and N10 are heavy.

Each node performs the load-balancing algorithm independently, and we choose N1 as an example to explain the load-balancing algorithm. N1 first queries the loads of N3, N6, N7, and N9 selected randomly from the system (Fig. 2b). Based on the samples, N1 estimates the ideal load T (i.e., $T=((L_{N1}+L_{N3}+L_{N6}+L_{N7}+L_{N9})/5)$. It notices that it is a light node. It then finds the heavy node it needs to request chunks. The heavy node is the most loaded node (i.e., N9) as N1 is the lightest among N1 and its sampled nodes {N3; N6; N7; N9}. N1 then sheds its load to its successor N2, departs from the system, and re-joins the system as the successor of N9. N1 allocates min $[L_{N9}- T_{N1}; T_{N1}]=T_{N1}$ chunks from N9.

In the example, N4 also performs the load rebalancing algorithm by first sampling {N3; N4; N5; N6; N7} (Fig. 2d).Similarly, N4 determines to re-join as the successor of N6. N4 then migrates its load to N5 and re-joins as the successor of N6 (Fig. 2e). N4 requests min $[L_{N6}-T_{N4}, T_{N4}] = L_{N6}-T_{N4}$chunks from N6.N6 is physically closer than N7 to N4.So, N4 re-joins as the successor of N6.

## IV.SECURITY

The data to be stored in the cloud is encrypted before storage for more security. The encryption is done by the key generated at the client side. Then the encrypted data is made into chunks and stored in various nodes.

When the server control performs operations on data like deletion or updating load imbalance problem occurs. This problem can be solved by the rebalancing algorithm which balances the load in the cloud after the above operations performed [16].

### *A. Encryption of data*

The data that is to be stored in cloud is not secure. In order to provide security to data, the data is stored in the encrypted form in the nodes. The file that is to be uploaded in the cloud is selected by the client. The encryption process is performed over the data through AES algorithm[15]. The encrypted file is made in chunks and stored in various nodes [16].

### *B. Splitting the data*

The encrypted file is partitioned into a number of chunks and is allocated in distinct nodes. 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 chunks of files are allocated uniformly among the nodes such that no node manages an excessive number of chunks [16].

### *C. Sending Data to Cloud*

As cloud is a centralized storage the data's have no security. So the encrypted file is made into chunks to provide more security to store the data in the cloud.

The splitted files are stored in the cloud and can be accessed from anywhere whenever needed. Thus storing a single file in various nodes has more security when compared to the file that is stored in a single node [16].

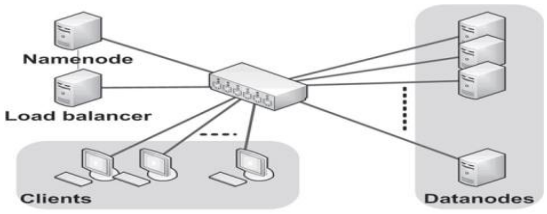
## V.SYSTEM ARCHITECTURE

The Experimental setup of our proposal is as shown in the below figure [1]:

Fig 3: the setup of the experimental environment

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 4, Issue 5, May 2015*




## VI.CONCLUSION

The overall goal of this paper is to make cloud computing effective by implementing the techniques that achieve the same. A novel load-balancing algorithm to deal with the load rebalancing problem in large-scale, dynamic, and distributed file systems in clouds has been presented in this paper. Our proposal strives to balance the loads of nodes and reduce the demanded movement cost as much as possible, while taking advantage of physical network locality and node heterogeneity. Particularly, our load-balancing algorithm exhibits a fast convergence rate. Load imbalance factor and algorithmic overhead are handled by developed algorithm efficiently. The data are stored in the cloud in the secure manner. The security for the data is provided by the encryption. The file that is to be uploaded in the cloud is encrypted by the key generated at the client side. For encryption we have used AES algorithm

## ACKNOWLEDGEMENT

We would like to thank our Guide **Mrs.Bhavya.G** Assistant Professor of Dept. of Information and Science and all of them who supported in completing this paper successfully.

## REFERENCES

[1] Hung-Chang Hsiao, Member, IEEE Computer Society, Hsueh-Yi Chung,Haiying Shen, Member, IEEE, and Yu-Chang Chao," Load Rebalancing for Distributed File Systems in Clouds" IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 24, NO. 5, MAY 2013
[2] 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.
[3] 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.
[4] Hadoop Distributed File System, http://hadoop.apache.org/hdfs/, 2012.
[5] VMware, http://www.vmware.com/, 2012.
[6] Xen, http://www.xen.org/, 2012.
[7] Apache Hadoop, http://hadoop.apache.org/, 2012.
[8] Hadoop Distributed File System "Rebalancing Blocks," http://developer.yahoo.com/hadoop/tutorial/module2.html#rebalancing,2012.
[9] K. McKusick and S. Quinlan, "GFS: Evolution on Fast-Forward,"Comm. ACM, vol. 53, no. 3, pp. 42-49, Jan. 2010.
[10] HDFS Federation,http:/hadoop.apache.org/ common/docs/r0.23.0/hadoop-yarn/hadoop-yarn-site/Federation.html, 2012.
[11] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H.Balakrishnan, "Chord: A Scalable Peer-to-PeerLookup Protocol for Internet Applications," IEEE/ACM Trans.Networking, vol. 11, no. 1, pp. 17-21, Feb. 2003.
[12] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed ObjectLocation and Routing for Large-Scale Peer-to-Peer Systems," Proc.IFIP/ACM Int'l Conf. Distributed Systems Platforms Heidelberg,pp. 161-172, Nov. 2001.
[13] 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.
[14] http://en.wikipedia.org/wiki/Gossip_protocol
[15] http://aesencryption.net
[16] R.Rajavadivu, A.BazilaBanu," Redistribution of Load in Cloud Using Improved Distributed Load Balancing Algorithm with Security"International Journal of Innovative Research in Science, Engineering and Technology *Volume 3, Special Issue 3, March 2014*
[17] Rajesh George Rajan, V.Jeyakrishnan, "A Survey on Load Balancing in Cloud Computing Environments" International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 12, December 2013