Load Rebalancing with Improved Security for Distributed File Systems in Cloud

Meera Suresh
M.Tech Scholar,
Dept. Computer Science
College of Engineering Munnar,
Kerala, India

Shine N. Das
Associate Professor,
Dept. Computer Science
College of Engineering Munnar,
Kerala, India

***Abstract*— This paper describes a Hadoop based load rebalancing algorithm along with a procedure for improving the security by using the cryptographic algorithms such as RSA and MD5. In a cloud computing environment, that contains large number of nodes, each of which having storage and processing capacities, failure is a norm and the nodes as well as the data may be upgraded, replaced, or added to the system resulting in load imbalance of the system. The load rebalancing algorithm can solve this issue. The security offered in cloud is very limited. So, for enhancing the security RSA and MD5 algorithms are incorporated to the system. Among the huge data those which need more security rather than just storing it are encrypted using RSA and checked periodically using MD5. Before encrypting using RSA the data is compressed, this improves speed of RSA. Replicas are stored in addition to recover from any distortion happening to data.**

Keywords*— Big Data; Cloud Computing; Hadoop; MD5; RSA

I. INTRODUCTION

Cloud Computing[1] offers easy access to high performance computing devices and storage infrastructure through web services. It provides massive scalability and reliability without sacrificing high performance. The cost associated with running an application in cloud depends on the amount of resources utilized by the application such as storage space and computational power. Cloud Computing refers to configuring, manipulating and accessing the applications online. It offers online infrastructure, data storage and application. Cloud Computing has numerous advantages, But the major issues associated with it such as security provided to the users cannot be ignored.

Cloud computing provides massive clusters for efficient large computation and data analysis. MapReduce[2] is a programming model which was first designed for improving the performance of large batch jobs on cloud computing systems. One of the most important performance bottlenecks in this model is due to the load imbalance between the reduce tasks. The input of the reduce tasks is known only after all the map tasks complete execution and the roles of the reducers are assigned beforehand, resulting in load imbalance between the reduce tasks. The objective is to examine the load rebalancing problem in cloud computing and to enhance distributed load rebalancing algorithm to cope with the load imbalance factor, movement cost, and algorithmic overhead [3]. Further investigation is done on the security provided in cloud and evaluates the Quality of Service-QOS i.e. the response of the whole system.

Section 2 describes the existing methods for load balancing and security in cloud. In Section 3, the proposed system is described. Further detailed study on the load imbalance problem is done in Section 4. Methodology and Results are given in later sections.

II. EXISTING SYSTEM

The load rebalancing algorithm proposed in [3] is an efficient method for balancing load in the distributed dynamic environment. The concepts mentioned in this paper are adopted in our system. However no method to solve security issues in cloud is included. In [4], describes a method for enhancing security by incorporating RSA to the system. Though using RSA is the secure way of keeping files, RSA is slow process. Here the point to be noted is that it is not practical to encrypt the entire big data. To solve this issue we have classified the data as critical data and data that needs to be stored alone. No method for checking the integrity of stored data is done in this existing method. Another method for encrypting data is prescribed in [5], where DES is used .The major drawback is that DES is a symmetric key system where the same key can be used for encryption and decryption. Here also no method is specified for integrity check.

III. PROPOSED SYSTEM

In cloud computing one server acts as the main server which can upload or download files in the connected sub servers. This method has single point failures in which the main server becomes performance bottle neck. To overcome this problem a fully distributed system on which the distributed load balancing algorithm that execute on every sub servers is proposed. The key building block for cloud computing application is Distributed File System that work based Map-Reduce programming model which is used for handling Big Data[6]. Every node in such system has storage and processing capacities. The load of a node is proportional to the number of file chunks stored by it. Balancing the load among the storage nodes is a critical function. In clouds the basic operation is splitting the file into chunks and allotting the chunks to the nodes where they will be processed. The node identifies themselves as under loaded or over loaded and pairing between the under loaded and over loaded nodes occurs, such that every node/server contains load which is within the specified limit called the average value.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

For dealing with the security issues RSA algorithm is used, which is an asymmetric cryptographic system having different keys for encryption and decryption. The sensitive data's will be compressed beforehand and will be encrypted using RSA i.e., for encryption the client's public key is used and the private key for decrypting the same will be send to the user's email id. So for further enhancing the security MD5 algorithm is used. Even if the data is attacked after encrypting, it can be found and will be reported to the user. The hash values of the data when it is first uploaded is found and it is stored in the data base .This will be compared with the hash values generated for the same data in periodic time intervals. If any change is detected it means that an attack have occurred and it will be reported. By compressing and encrypting data, it can be transmitted over unsecure and bandwidth constrained channel. Compressing the data has other notable advantages such as decreased movement cost and space consumption which improves the total system efficiency and speed of RSA.

The distributed file systems like the Google GFS and Hadoop HDFS in clouds rely on central nodes to handle the metadata information of the file systems and also to balance the loads of storage nodes based on that metadata. Although this approach simplifies the design and implementation, recent studies concludes that when the number of files, number of storage nodes and the number of accesses to files increase linearly, the central nodes (e.g., the master node in Google GFS) become a performance bottleneck. In the proposed system the dependence on central nodes are completely avoided. The storage nodes are structured as a network based on the distributed hash tables. DHT's enables nodes to repair and self organize by constantly providing look up functionality in the node dynamism and management. A unique identifier is provided to each file chunk such that finding a file chunk means a rapid key look up. The proposed algorithm outperforms the existing load balancing approaches in terms of imbalance factor, algorithmic overhead and movement cost. Load-balancing algorithms based on DHTs are extensively studied (e.g., [7-13]), but most existing solutions are designed without taking into account factors such as movement cost, node heterogeneity and physical network locality in the reallocation of file chunks [14-15].Our proposed system considers all three and also considers the security issues in cloud thereby improving the overall system performance. The architecture of the proposed system is given in Fig 1.

I. THE LOAD REBALANCING PROBLEM

The chunk servers are organized as a DHT network in our proposal; that is, each chunk server implements a DHT protocol such as Chord [12] or Pastry [13]. A file in the system is partitioned into a number of fixed-size chunks, and "each "chunk

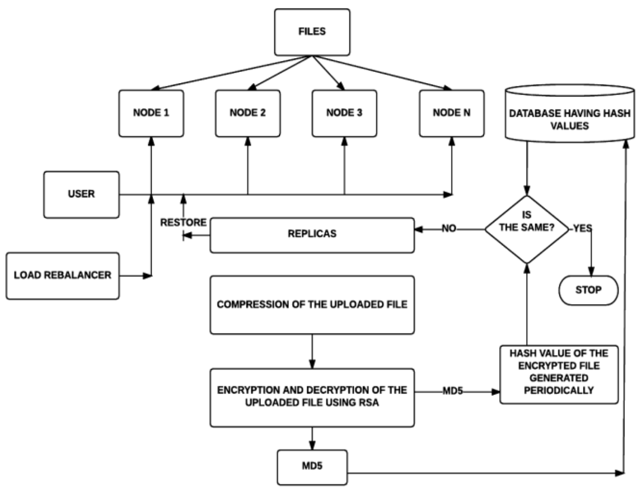

FILES
NODE 1
NODE 2
NODE 3
NODE N
DATABASE HAVING HASH VALUES
USER
RESTORE
REPLICAS
NO
IS THE SAME?
YES
LOAD REBALANCER
STOP
COMPRESSION OF THE UPLOADED FILE
HASH VALUE OF THE ENCRYPTED FILE GENERATED PERIODICALLY
ENCRYPTION AND DECRYPTION OF THE UPLOADED FILE USING RSA
MD5
MD5

Fig 1.System architecture

has a unique chunk identifier named with a globally known hash function such as SHA1. Each chunk server is also given a unique ID. The number of files can be hundred, thousand or even more. The chunk servers are represented as V. For short, the n chunk servers are denoted as $1,2,3..n$. The successor of chunk server i is the chunk server $i+1$ and the successor of chunk server n will be chunk server 1. In a typical DHT, a chunk server i host the file chunks or data items whose ID is greater than or equal to the ID of the node. DHTs is used in our proposed system as it is having the following advantages:

1) The chunk servers will self-configure and self-heal in our proposal because of their arrivals, failures and departure, simplifying the system provisioning and management.
2) If a node leaves, then its locally hosted chunks are reliably migrated to its successor. if a node joins, then it allocates the chunks whose IDs immediately precede the joining node from its successor to manage.

A large-scale distributed file system having a set of chunk servers V in a cloud, where the cardinality of V is n. n can be 1000, 10,000 or even more. Number of files is stored in the n servers. The set of files is taken as F .The subset of F is f , each file is partitioned into fixed size disjoint chunks C_f .Aim is to avoid the load imbalance problem while reducing the movement cost as much as possible.

Movement cost is the number of file chunks that should be moved to balance the loads in the file servers. The average load that a node can have is found as:

$$A = \frac{\sum_{f \in F} C_f}{n} \qquad (1)$$

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

Then, our load rebalancing algorithm aims to minimize the load imbalance factor in each chunk server i as follows:

$$\|L_i - A\| \qquad (2)$$

Where, L_i represents the present load of node i .The load imbalance problem is an NP hard problem. It is considered as a version of Knapsack problem. Every node is identified as under loaded/light node or as over loaded/heavy node using parameters ΔL and ΔU . A node is *light loaded* if the number of chunks it hosts is smaller than the threshold of $(1-\Delta L)A$ (where $0 \le \Delta L < 1$) and a node is *heavy loaded* if the number of chunks it hosts is greater than the threshold of $(1+\Delta U)A$ (where $0 \le \Delta U < 1$).

Fig 2. Gives an illustration of the load imbalance problem. where (a) Four nodes having same capacity but different load (b) Nodes in the balanced state.

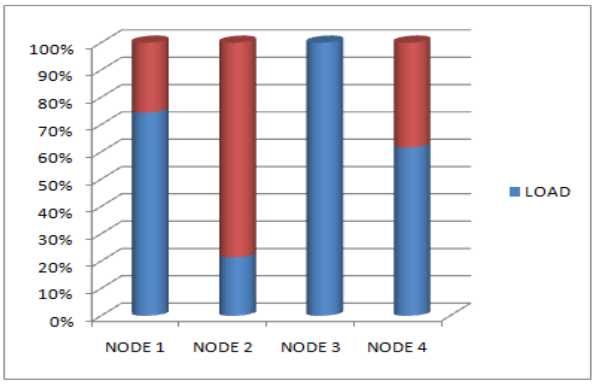
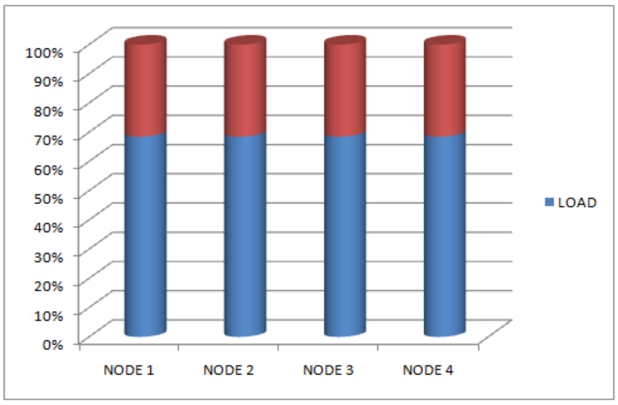


Fig 2. (a) An example for load imbalance problem



(b) Nodes in the balanced state

II. METHODOLOGY

The proposed system is implemented by using the Secure_Load_Rebalance algorithm to which the concepts of physical network locality, Node heterogeneity and managing replicas are added. Along with this for security the method of *compression and encryption* is done.RSA algorithm is executed on the compressed data for this. For further enhancing the security of data integrity checking is done by using MD5 algorithm.

A. *Secure_Load_Rebalance algorithm:*

1. Initialize server and its sub servers.
2. Establish connection between sub-server and servers using the IP or Port number.
3. Upload File to server that should be shared.
4. If the File is specified to be encrypted, make replica of it.
5. Split the file into multiple chunks.
6. Find the hash value of each chunk by using MD5.
7. Calculate the each sub server memory.
8. Divide the total chunks value by total number of sub servers.
9. Upload each chunk into sub servers after encrypting it using RSA based on its memory capacity.
10. If Capacity is less then transfer the excess chunks into next Sub-servers.
11. Each chunk will be appended with an index value.
12. When the client ask for a file, which will be conventional from dissimilar sub-servers based on the index value.
13. Client collects all the chunks then the file will be decrypted, after that so will be view by user.
14. Perform integrity checking in regular intervals of time to find whether any attack has occurred by comparing the hash values generated in each check with the initial hash value of the chunk.
15. If an attack occurred report it and replace the damaged chunk with the replica of original. Further the concepts such as physical network, Heterogeneity of nodes, managing replicas and cryptographic techniques for security enhancement is also considered.

B. *Exploiting Physical Network Locality*

A Distributed Hash Table network is an overlay on the application level. The logical closeness of the nodes derived from the DHT does not necessarily match the physical closeness information in reality. That means a message or data travelling between two nodes in a DHT overlay may travel a long physical distance through several physical network links, when it should take the shortest path. If it takes the longest path it causes high movement cost and consumption of network resources will also be very large. In order to avoid these issues the *Migrate locality aware algorithm* is added to the Secure Load Rebalancing Algorithm in the proposed system so that data migration occurs between the nearest nodes.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

```
Migrate_Locality_Aware $(i,V)$:

A light node $i$ join a heavy node $j$ which is physically closest to $i$.

Input: a light node i and set of chunk servers $\{V_1, V_2 \ldots V_{nv}\}$

1. $C \leftarrow \phi$
2. For $K = 1$ to $nv$
3. $C \leftarrow C \cup Secure\_Load\_Rebalance;$
4. $j \leftarrow$ The node in $C$ physically closest to $i$
5. MIGRATE $(i, j)$
```

C. *Taking Advantage of Node Heterogeneity*

When such a migration is done, heterogeneity of the system should be considered. Different nodes in the system can have different capacities in the distributed environment. The idea that load of a node should be proportional to its capacity is implemented in the proposed system. Consider capacity of nodes $(C_1, C_2 .. C_n)$ each node consists of estimated number of file section. Load balancing considering the capacity of each node is done as follows.

$$A_i = \gamma C_i \qquad (3)$$

Where γ is the weight per unit capacity of node and it is calculated as follows:

$$\gamma = \frac{m}{\sum_{k=1}^{n} C_k} \qquad (4)$$

Where m is the total file chunks stored in a system.

D. *Managing Replicas*

A replica of the sensitive data is stored in other nodes. Even if node failure occurs the data will never be lost. The replica chunks also hold a unique ID with which the original data can be recovered. After the integrity check by MD5 if an attack is reported ,the original data will be recovered from the replica.

III. RSA

RSA is an asymmetric algorithm introduced by Ronald Rivest, Adi Shamir and Leonard Adlemanin in 1977.RSA is based on a one-way function called integer factorization in number theory. It is a block cipher in which every message is mapped to an integer[16]. The main benefit in using RSA is that it uses different keys for encryption and decryption. But the speed of RSA is very low so for overcoming this, the data will be compressed before it undergo cryptographic operations. The public key of the user is used for encryption and for decryption private key is used. When the data is uploaded it will be encrypted at the server side and decryption is done by the Cloud user. Once the data is encrypted with the Public-Key, it can be decrypted with the corresponding Private-Key only. In our system this key is mailed to user specified e-mail id.

RSA algorithm involves three steps.

(i) First, in Key generation before the data is encrypted, Key Generation should be done. This process is done between the Cloud service provider and the user.
(ii) Second, in Encryption is the process of converting original plain text (data) into cipher text (data).
(iii)Third, Decryption is the process of converting the cipher text (data) to the original plain text (data).

Key Generation Algorithm:

1) Randomly and secretly choose two large primes suppose p, q and
Compute n = p. q
2) Compute ϕ (n) = (p - 1) (q - 1).
3) A Random Integer e is selected such that 1< e< n and gcd (e, ϕ) = 1 .
4) Compute d such as e. d $\equiv$ 1 mod ϕ (n) and 1 < d < ϕ (n).
5) Public Key: (e, n)
6) Private Key: (d, n).
7) This key sends to user's email id.

Encryption Process:
1) When the data is uploaded by user an specifies that it should be encrypted then the data encryption is done as follows
2) Cipher text c = message e mod n

Decryption Process:
1) Plain text p = cipher text d mod n.
2) User can get the original data by decrypting it using the private key.

IV. MD5

MD5 is used for checking data integrity. It generates a fingerprint of the data [17]. If the fingerprint of the data in the initial state and the fingerprint generated for the same data in the future are same it means that the data is not hacked, else it means that the data is been attacked. The fingerprint or hash value is 128 bit and it is usually expressed as 32 bit hexadecimal number. In our system the sensitive data will be integrity checked using MD5 at regular intervals .If it is found to be attacked, then the damaged block will be replaced using its replica. We choose MD5 as it is faster, which is most important feature required when handling huge amount of data.

V. QOS EVALUATION

Quality of Service (QoS) represents the overall experience a user or application will receive over a network. The QoS provided to the user is improved by uniformly distributing data avoiding data loss and also by reducing the response time or by reducing time consistency of the system. Although the speed at which RSA works is very low, after compressing the data this can be improved. Although it is not practical to compress the entire data or Big Data since we are compressing the sensitive data the storage space utilization and computation power utilized by RSA can be considerably reduced.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

The users can upload data, specify whether it needs to be encrypted, download it by using the secret key send to the user specified email id's and also can check whether any sort of attack has happened to the data. If the data is reported as distorted it can be replaced from the replicas.These are shown in Fig 3. (a). Shows the System features.(b).Specifying the data as critical so that it can be encrypted. (c).Verifying the data using MD5.(d).Replacing the data from replicas if data is distorted.

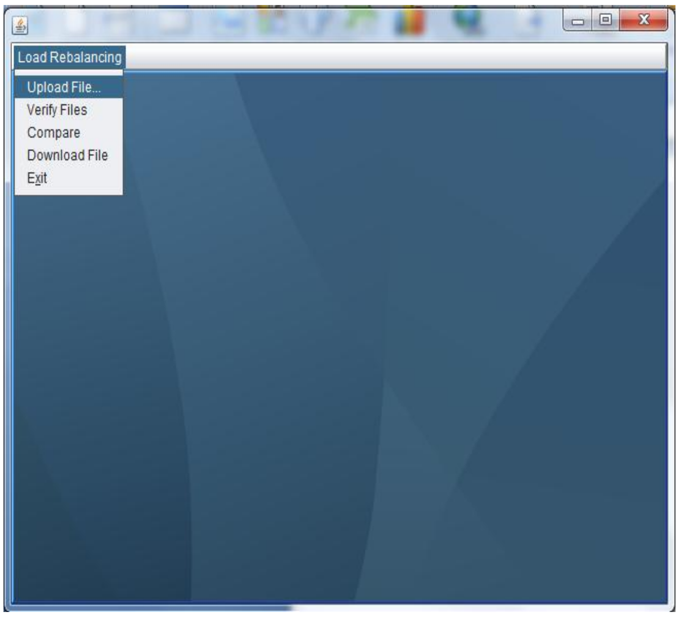
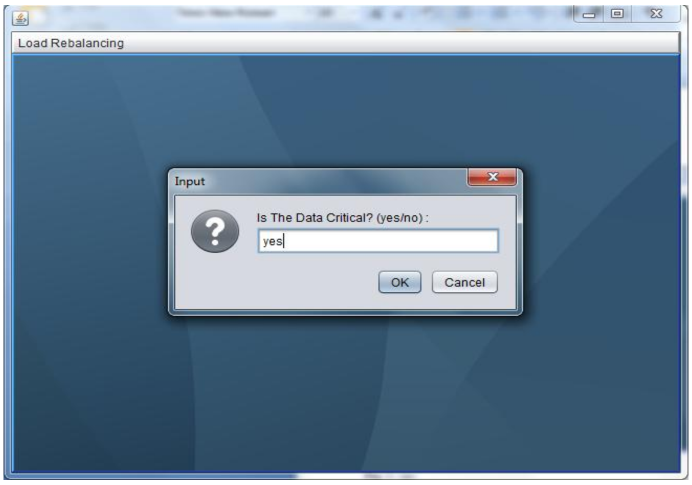
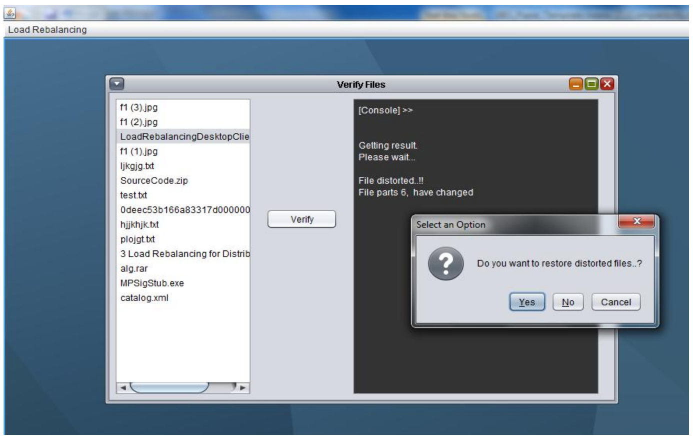


Fig 3. (a) System features



Is The Data Critical? (yes/no) :
yes

Fig 3.(b) Specifying a data as critical or confidential

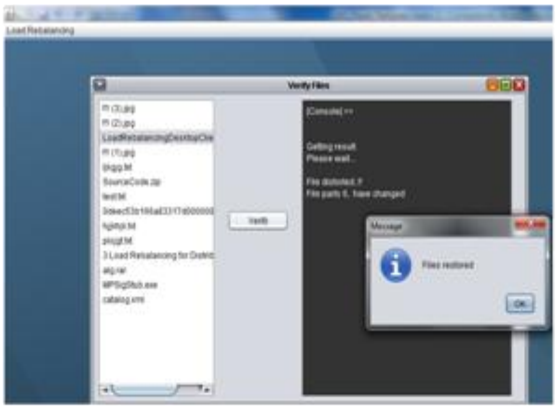


f1 (3).jpg
f1 (2).jpg
LoadRebalancingDesktopClie
f1 (1).jpg
ljkgjg.txt
SourceCode.zip
test.txt
0deec53b166a83317d000000
hjjkhjk.txt
plojgt.txt
3 Load Rebalancing for Distrib
alg.rar
MPSigStub.exe
catalog.xml

```
[Console] >>

Getting result.
Please wait...

File distorted..!!
File parts 6,  have changed
```

Do you want to restore distorted files..?

Fig 3.(c)Verifying using MD5



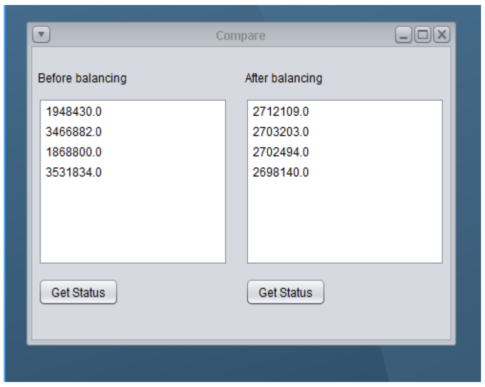
```
[Console] >>

Getting result
Please wait...

File distorted..!!
File parts 6,  have changed
```

Files restored

Fig 3.(d) Replacing the distorted data by using replicas

VI. Result And Discussion

Fig4.(a) and (b) shows the simulation results of load distribution before and after executing the proposed load rebalancing algorithms with the compression and security feature. The proposed system is compared against a centralized approach in a production system which uniformly distributes across sub servers. It outperforms the previous distributed algorithmic rule in terms of load imbalance issue, movement value, and recursive overhead. Our proposal is implemented in the Hadoop Distributed File System.



Before balancing	After balancing
1948430.0	2712109.0
3466882.0	2703203.0
1868800.0	2702494.0
3531834.0	2698140.0

Fig 4. (a) Test result

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

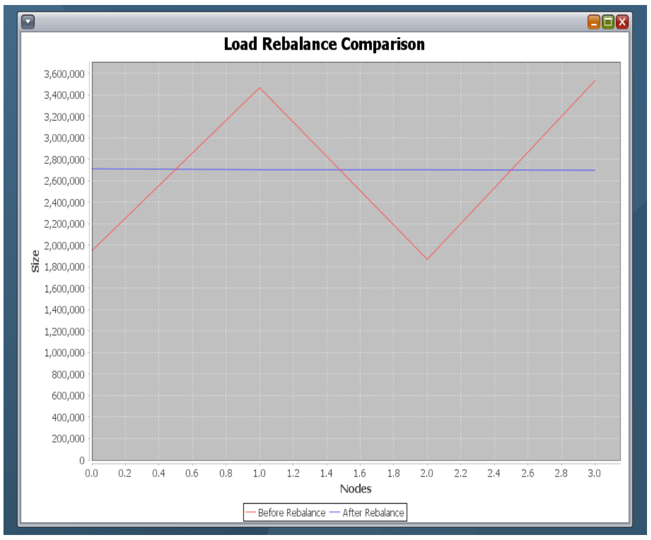


Fig 4. (b)Test result-graphical representation.

VII. Conclusion

A cloud computing application works based on the Map-Reduce programming paradigm. The major issue that occurs in distributed environment is the load imbalance between the nodes or servers. If all the nodes in the system can be balanced then the system performance can be improved by the maximum utilization of bandwidth available. For balancing the factors such as movement cost, algorithmic overhead, power consumption and network resource utilization should be considered. Minimizing all these factors can improve the overall system performance. Another major issue in the cloud environment is the security. The proposed system offers a combined measure for overcoming both these major issues in the distributed cloud environment. The major application of this technique is with Big Data handling .

References

[1]. http://www.ibm.com/cloud-computing/us/en/what-is-cloud computing. html, "what is cloud computing,"

[2]. G. Yang, "The application of map reduce in the cloud computing," in Intelligence Information Processing and Trusted Computing (IPTC), 2011 2nd International Symposium on, pp. 154-156, IEEE, 2011.

[3]. H.-C. Hsiao, H.-Y. Chung, H. Shen, and Y.-C. Chao, "Load rebalancing for distributed file systems in clouds," Parallel and Distributed Systems, IEEE Transactions on, vol. 24, no. 5, pp. 951-962, 2013.

[4]. Kokilavani .K, Department Of Pervasive Computing Technology, Kings College Of Engineering, Punalkulam, Tamil nadu "Enhance load balancing algorithm for distributed file system in cloud" International Journal of Engineering and Innovative Technology (IJEIT) Volume 3, Issue 6, December 2013.

[5]. A. Sumanth, B.R.Singh, M.Sangeetha "Secure and Privacy-Preserving Distributed File Systems on Load Rebalancing in Cloud Computing" International Journal of Computer Trends and Technology (IJCTT) – Volume 18 Number 4 – Dec 2014

[6]. A. S. Sabia and G. N. D. U. Arora Department of Computer Science Engineering, "Technologies to handle big data,".

[7]. M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip- based aggregation in large dynamic networks," ACM Transactions on Computer Systems (TOCS), vol. 23, no. 3, pp. 219-252, 2005.

[8]. M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, pp. 551-556, IEEE, 2010.

[9]. S. Penmatsa and A. T. Chronopoulos, "Game-theoretic static load balancing for distributed systems," Journal of Parallel and Distributed Computing,vol. 71, no. 4, pp. 537- 555, 2011.

[10]. A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I.Stoica, "Load balancing in structured p2p systems," in Peer to-Peer Systems II, pp. 68-79, Springer, 2003.

[11]. B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp,and I. Stoica, "Load balancing in dynamic structured p2p systems," in INFOCOM 2004. Twentythird AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 4, pp. 2253-2262, IEEE, 2004.

[12]. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," ACM SIGCOMM Computer Communication Review, vol. 31, no. 4, pp. 149-160, 2001.

[13]. A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer- to-peer systems," in Middleware 2001, pp. 329-350, Springer, 2001.

[14]. H. Shen and C.-Z. Xu, "Locality-aware and churn- resilient load-balancing algorithms in structured peer-to-peer networks," Parallel and Distributed Systems, IEEE Transactions on, vol. 18, no. 6, pp. 849-862, 2007.

[15]. Y. Zhu and Y. Hu, "Efficient, proximity-aware load balancing for dht-based p2p systems," Parallel and Distributed Systems, IEEE Transactions on, vol. 16, no. 4, pp. 349-361, 2005.

[16]. R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM,vol. 21, no. 2, pp. 120-126, 1978.

[17]. R. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, MIT LCS & RSA Data Security, Inc., April 1992.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)