ISSN:0975-9646

# Design and Implementation of Performance Guaranteed Symmetric Load Balancing Algorithm

Shaik Nagoor Meeravali#1, R. Daniel*2, CH. Srinivasa Reddy#3

*#M.Tech, Department of Information Technology, Vignan's Institute of Information Technology, JNTU-KAKINADA, Andhra Pradesh, India*

**Associate Professor*

*Department of Information Technology, Vignan's Institute of Information Technology, JNTU-KAKINADA, Andhra Pradesh, India*

*Abstract*— **Mainly Distributed Hash Table (DHT) uses decentralized load balance algorithms which are based on virtual servers for participating in asymmetric peers. Require the participating peers to be asymmetric, there by introducing another load imbalance problem which is symmetric and promise no precise performance metrics. In this paper, an original symmetric load balancing algorithm for DHTs is introduced where the peers approximate the system state with histograms. Unlike other algorithms, proposed work guarantees analytical performance in terms of the load balance factor and high convergence rate. Through implementation using Java and SQL server , shown that proposal work performs better in terms of load balance factor with a comparable cost.**

*Keywords*— **DHT, Symmetric, Asymmetric, Load Balance, NAT, P2P, and Virtual Server.**

## I. Introduction

Load balancing is a technique used to spread a network service workload between two or more devices. Benefits include scalability, reliability, efficiency, redundancy, and minimized response time. Load balancing can be achieved either by running an application on a server (software load balancing) or by using a special purpose device (hardware load balancing). Software load balancing applications use a server's CPU to process requests. A hardware load balancer uses a processor made specifically for this purpose to handle more requests. The load balancer brokers connections and distributes the load between the client and the servers. The server that receives the connection is chosen using a preset algorithm. The packets that make up the client request are translated by the load balancer (using a process called NAT-Network Address Translation) before being sent to the server. The client and the server are equally unaware of the load balancer. Mainly Distributed Hash Table uses decentralized load balance algorithms which are based on virtual servers for participating in asymmetric peers. Require the participating peers to be asymmetric, thereby introducing another load imbalance problem which is symmetric and promise no precise performance metrics. In this paper, an original symmetric load balancing algorithm for DHTs is introduced where the peers approximate the system state with histograms. Proposal work guarantees analytical performance in terms of the load balance factor and high convergence rate. Through implementation using java and SQl server, shown that the proposal work performs better in terms of load balance factor with a comparable cost. Distributed hash tables (DHTs)[1] are key building blocks in the design and implementation of successful distributed applications. Designing a load-balanced, heterogeneity-aware DHT with virtual servers is technically challenging. In particular, load balancing algorithms[5] designed for DHTs based on virtual servers need to take the following into consideration.

**1. Load balance and movement cost**: By load balance, this mean that each peer manages the load proportional to its capacity. Previous studies[6] suggest migrating virtual servers among the participating peers in order to balance peer load. However, this is at the expense of introducing movement cost due to the migration of virtual servers. How to balance peer load while reducing movement cost as much as possible thus is a critical issue.

**2. System dynamics**: Load balancing algorithms need to bear the system dynamics in mind because nodes may dynamically join and leave DHTs. In addition, the load of a virtual server may change from time to time, aggravating the load imbalance problem in the DHTs.

**3. Algorithmic robustness and workload**: Load balancing algorithms need to be robust without introducing the performance bottleneck and the single point of failure. In addition, as load balancing algorithms[5] incur algorithmic workloads, such workloads shall not induce another load imbalance problem. On the other hand, a well-designed load balancing algorithm will not generate considerable overheads.

**4. Performance guarantee:** Load balancing algorithms shall work well with performance guarantee, given any system instance. Specifically, DHT networks may operate in dynamic and large-scale environments, thus presenting a large number of problem instances for performance investigation.

## II. Related Work

Earlier studies [12], [13], [14], [15] have proposed load balancing algorithms, targeting static, small-scale, and/or homogeneous environments. Due to space limitation, we provide a concise review of the load balancing techniques designed for DHTs in this section. As the previous study [16] has provided a survey for the load sharing algorithms in traditional high-performance computing systems, we refer interested readers to [17], [18] for a survey on the load balancing algorithms in DHTs.

In contrast to evenly partitioning the number of objects and thus the key space to each participating peer, Chord

suggests the notion of virtual servers to balance the loads of the peers [1] further. Existing works proposing load balancing algorithms for DHTs with virtual servers can be found in the literature [6], [9], [10], [11]. The many-to-many framework presented in [6] categories participating peers into light and heavy nodes. The light and heavy peers register their load with some dedicated nodes, i.e., the directories. As noted by the authors in [7], the many-to-many framework essentially reduces the load balancing problem to a centralized algorithmic problem. As the entire system heavily depends on the directory nodes, the directory nodes may thus become the performance bottleneck and the single point of failure.
In contrast, in this paper, we are particularly interested in obtaining fully performance guaranteed solutions using symmetric manner to the load balancing problem.

## III. SYSTEM ARCHITECTURE

Fig.1 shows the system architecture of symmetric load balancing which gives the conceptual model that defines the structure, behaviour, and more views of a system.

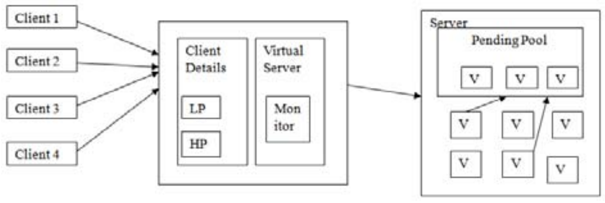


**Fig. 1 Symmetric Load Balancing Algorithm**

**Client:** In this system the Client sends the services to Server which is manage the load using the Virtual server.DHT categorised into two categories of peers that are Light Peers (LP) and Heavy Peers (HP).
**Server:** Server manages the load by creating virtual servers. These servers maintain the pending pool for requested data. If the Virtual Server lost, the lost virtual server has been handled by the Pending Pool and the requested data is getting from the Pending Pool. Processes of pending pool are shown in Fig.2.

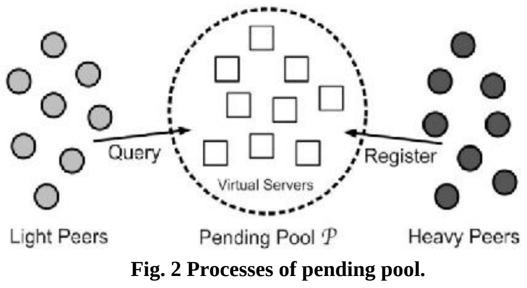


**Fig. 2 Processes of pending pool.**

## IV. MODULES OF THE SYSTEM

There are 5 modules in this proposed system
1. **User Interface Design**
2. **Client Sever communication**
3. **DHT Implementation**
4. **Virtual Server Implementation**
5. **Pending Pool Implementation**

### User interface design

In Fig. 3 shows the user interface module design which consists of windows. These windows are used to send a message from one peer to another. We use the Swing package available in Java to design the User Interface. Swing is a widget toolkit for Java. It is part of Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs. In this module mainly, focusing the Client Home Page with the Partial knowledge information. Distribute Hash Table maintain a neighbour information and it gets the partial information about the clients and virtual server. Like client name, ip address, port no etc. With this information client can communicate with other clients in the network and share their data.

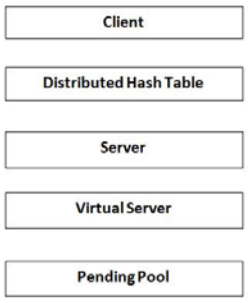


**Fig 3. User Interface Design**

### Client Sever communication

In this Module each client has been store the files in Server . During the time of storage may be load appeared in the Server. Accessing the Client may be increased then the Server Performance is reduced. Server has been handling the load using the virtual server. Virtual Server has been automatically created during the time of load in server which client access level increased. The detailed architecture this module is shown in Fig.4

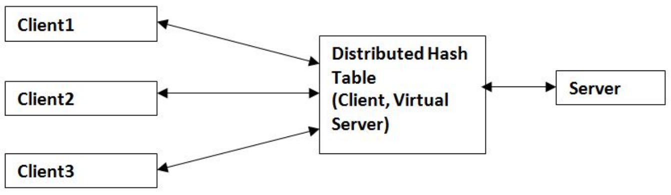


**Fig. 4 Client Server communication**

### DHT Implementation

The DHT implementation shown in Fig.5, which shows the Distributed Hash Table maintains two values such as a Client details and Virtual Server details.DHT has been categories the client into two category based upon the Storage file size, first one is Light peer and second one is Heavy Peer.DHT have been only monitor the which one is Light Peer and Heavy Peer. Then the Client requested files have been stored in a Virtual server using the Distributed Hash Table .Client has been give the request a file to Virtual Server .DHT has been check which Virtual Server maintain the Client file and forward to Virtual Server. Distributed Hash Table has been reducing the Movement cost in a network. In a typical DHT, participating nodes can join and leave, arbitrarily. Thus, the reallocation of a virtual server from a source peer to a destination peer can be simply done by simulating the leave and join operations offered by a typical DHT.

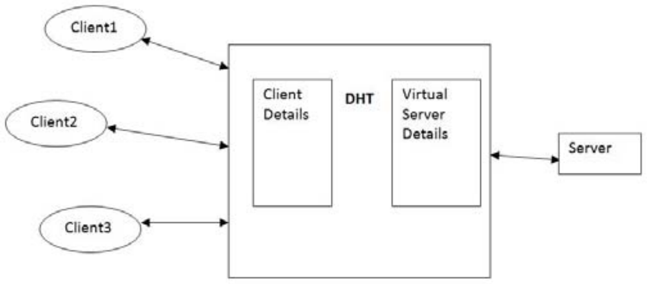


**Fig. 5 DHT Implementation**

### Virtual Server Implementation

Virtual Server has been storing the all client files. Each Virtual Server has a separate memory. Client has been give the request file to Virtual Server.DHT has been check which Virtual Server is free memory to store the files. That is maintain the Client file and forward to Virtual Server. The Virtual Server has been Send the Client requested file to Client. Virtual Server has been maintaining the uploaded client files only. In case it is lost, all files are moved to the Pending pool. The detailed implementation of this module is in Fig.6

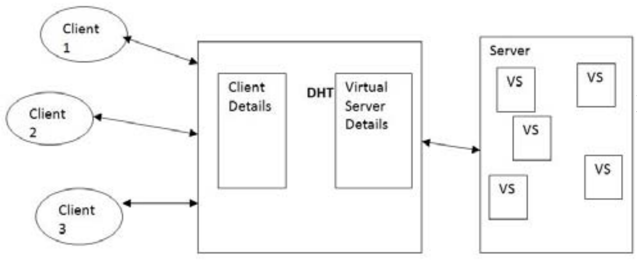


**Fig. 6 Virtual Server Implementation**

## V. METHODOLOGY

Symmetric Load balancing algorithm which is one of load balancing technique which is used for Distributed Hash Table to balance their virtual server load in a network. The Load balance factor has been minimizing the movement cost in network.

### Load balance factor

Given the distribution of capacities of the peers and loads of the virtual servers.

### Movement cost

Given the set of peers, set of virtual servers, peer i migrates a subset of its virtual servers to other peers.

### A Symmetric Load Balancing Algorithm:

**Start:** load balancing, f(N,V).

**Input:** N be the set of participating peers in a DHT.
V be the set of virtual servers deployed over N.

**Output:** Finally an error correction is observed for the allocated peers and their virtual servers.

1. Initialize each peer i Є N has a maximum capacity of Ci max and hosts a set of virtual servers Vi is sub set of V.
2. Vi intersection Vj = φ; for any into equal to j Є N.
3. Each virtual server v Є Vi has a load denoted by Lv
4. A to reallocate and balance the loads among the
5. participating peers, y peer i manages the total load of virtual servers proportional to its Ci max A I
6. A computes a subset Vi subset of V for each peer i, such that the following equation is minimized:

To ease our discussion, we define the following terminologies and notations:

$$\sum_{v \in V_i^A} L_v - \frac{\sum_{v \in V} L_v}{\sum_{k \in N} c_k^{max}} \times c_k^{max}$$

STEP1: The load per unit capacity, which is a peer that hosts in a load-balanced DHT, is defined as

$$\mu \triangleq \frac{\sum_{v \in V} L_v}{\sum_{k \in N} c_k^{max}}$$

STEP2: The ideal load, denoted by , which peer i 2 N manages in a load-balanced
DHT, is

$$I_i \triangleq \mu\, c_i^{max}$$

STEP3: The remaining capacity of peer is

$$c_i \triangleq I_i - \sum_{v \in V_i} L_v$$

In load balancing algorithm intends to balance loads of participating peers by minimizing. It also aims to reduce the movement cost as much as possible.

- In a typical DHT, participating nodes can join and leave, arbitrarily. Thus, the reallocation of a virtual server from a source peer to a destination peer can be simply done by simulating the leave and join operations offered by a typical DHT.
- The load of any virtual server v at a particular time is the sum of loads of objects hosted by v at that time; the load of a peer i is the aggregate of loads of virtual servers maintained by i.
- The potential metrics for measuring the loads include CPU utilization, storage space, etc.
- This is able to calculate the difference between light peers and heavy peers.
- Based on above step it will mitigate the virtual servers has to be created in pending pool or it has to be assigned on the light peers.
- Now this will calculate the probability distribution for each peer to which the virtual server is allocated from pending pool.
- Finally an error correction is observed for the allocated peers and their virtual servers.

  In proposed work, as each heavy peer selects its virtual servers with small sizes to migrate, the resultant movement cost is small. Thus, analyzing the load balance factor for each peer suffices. The load balance factor of peer I (denoted by LBFI) is defined as follows:

$$LBF_i \triangleq \frac{\sum_{v\in V_i^A} L_v}{c_i^{max}}$$

Here represents load balancing equation,

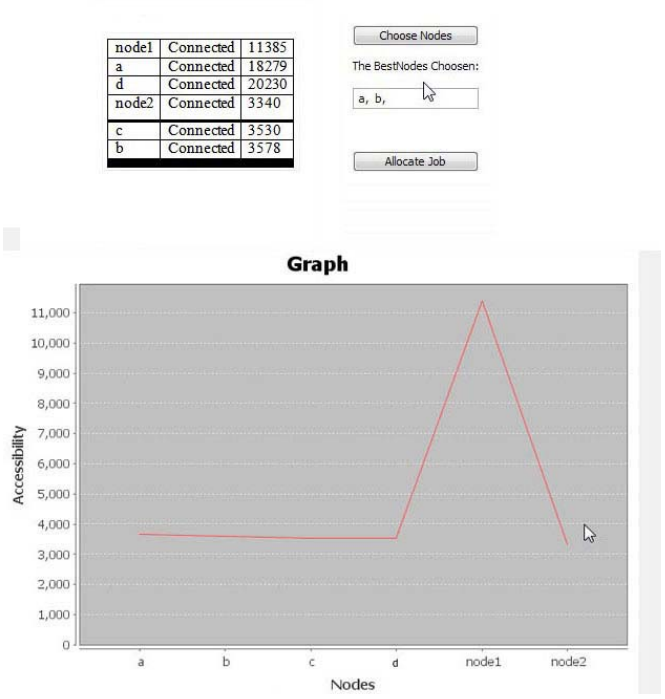
$$\sum_{v\in V_i^A} L_v - \frac{\sum_{v\in V} L_v}{\sum_{k\in N} c_k^{max}} \times c_i^{max}$$

$$c_i^{max}\left(\frac{\sum_{v\in V_i^A} L_v}{c_i^{max}} - \mu\right)$$

$$c_i^{max}\left(LBF_i - \mu\right)$$

## VI. Results

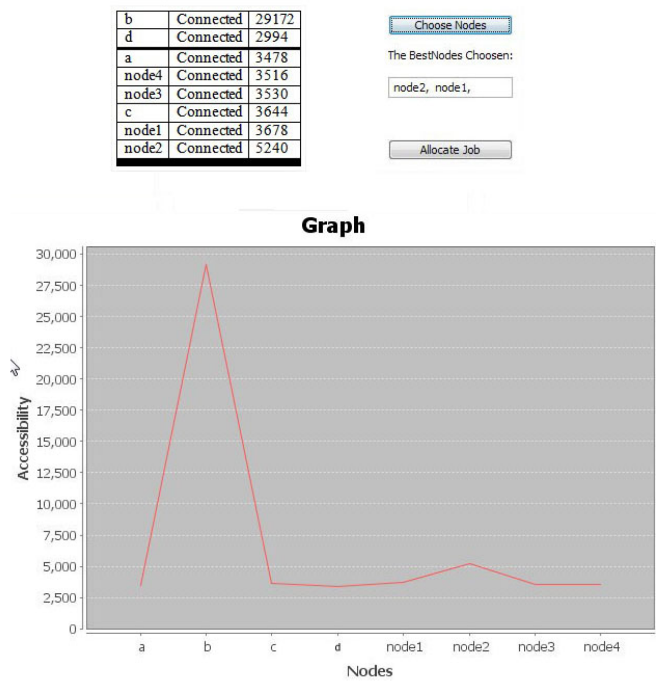
Symmetric load balancing algorithm implemented using java and SQl server, shows that this performs better in terms of load balance factor with a comparable cost. Distributed hash tables (DHTs) are key building blocks in the design and implementation of successful distributed applications.



| | | |
|---|---|---|
| node1 | Connected | 11385 |
| a | Connected | 18279 |
| d | Connected | 20230 |
| node2 | Connected | 3340 |
| c | Connected | 3530 |
| b | Connected | 3578 |

**Fig.7 Load balancing graph for 7 nodes**

In Fig.7 shows, there is a less work load at 'a' and 'b'. So the work can be assign these peers only to assign the work. The remaining nodes has already heavy work load, so they can't share work.



| | | |
|---|---|---|
| b | Connected | 29172 |
| d | Connected | 2994 |
| a | Connected | 3478 |
| node4 | Connected | 3516 |
| node3 | Connected | 3530 |
| c | Connected | 3644 |
| node1 | Connected | 3678 |
| node2 | Connected | 5240 |

**Fig.8 Load balancing graph for 9 nodes**

In Fig.8, there are two peers with less load work that are “node1” and “node2”. So the new work can be assigned to these two peers only. The implemented results shown that if the numbers of clients nodes increase then parallel number of virtual serves also increasing to balance the work load. The details of comparison of different nodes sharing of work load using symmetric load balancing algorithm shown in Fig.7 and Fig.8

## VII. Conclusion

In this paper, implemented a load balancing algorithm for the reallocation of virtual servers in DHTs. This load balancing algorithm operates in a fully decentralized manner by having each participating peer estimate the probability distribution of loads of virtual servers selected for migration and the probability distribution of the remaining capacities of under-loaded peers. Network performance analysis is a follow-up to other monitoring and tuning efforts that are specific to a work station or server computer.

## References

[1] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications,” IEEE/ACM Trans. Networking, vol. 11, no. 1, pp. 17-21, Feb. 2003.

[2] 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, pp. 161-172, Nov. 2001.

[3] 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.

[4] BitTorrent, http://www.bittorrent.org/index.html, 2012.

[5] J. Stribling, E. Sit, M.F. Kaashoek, J. Li, and R. Morris, “Don’t Give Up on Distributed File Systems,” Proc. Sixth Int’l Workshop Peer-to-Peer Systems (IPTPS ’07), Feb. 2007.

[6] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS '02), pp. 68-79, Feb. 2003.

[7] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," Performance Evaluation, vol. 63, no. 6, pp. 217-240, Mar. 2006.

[8] C. Chen and K.-C. Tsai, "The Server Reassignment Problem for Load Balancing in Structured P2P Systems," IEEE Trans. Parallel Distributed Systems, vol. 12, no. 2, pp. 234-246, Feb. 2008.

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

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

[11] 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.

[12] H.-C. Lin and C.S. Raghavendra, "A Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC)," IEEE Trans. Software Eng., vol. 18, no. 2, pp. 148-158, Feb. 1992.

[13] F.C.H. Lin and R.M. Keller, "The Gradient Model Load Balancing Method," IEEE Trans. Software Eng., vol. 13, no. 1, pp. 32-38, Jan. 1987.

[14] L.M. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," IEEE Trans. Software Eng., vol. 11, no. 5, pp. 491-496, May 1985.

[15] L.M. Ni, C.-W. Xu, and T.B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," IEEE Trans. Software Eng., vol. 11, no. 10, pp. 1153-1161, Oct. 1985.

[16] T.L. Casavant and J.G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," IEEE Trans. Software Eng., vol. 14, no. 2, pp. 141-154, Feb. 1988.

[17] Y. Zhu, "Load Balancing in Structured P2P Networks," Handbook of Peer-to-Peer Networking, Springer, July 2009.

[18] H. Shen, "Load Balancing in Peer-to-Peer Systems," Handbook of Research on Scalable Computing Technologies, IGI Global, July 2009.

[19] A. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," Proc. ACM SIGCOMM '04, pp. 353-366, Aug. 2004.

[20] P. Ganesan, M. Bawa, and H. Garcia-Molina, "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems," Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04), pp. 444-455, Sept. 2004.

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

[22] K. Kenthapadi and G.S. Manku, "Decentralized Algorithms Using Both Local and Random Probes for P2P Load Balancing," Proc. 17th ACM Symp. Parallel Algorithms and Architectures (SPAA '05), pp. 135-144, July 2005.