ISSN:0975-9646

Load Balancing In Structured Peer to Peer Systems

Mr. Sumit A. Hirve, Dr. Suhas H. Patil,

Department of Computer Engineering, Bharati Vidyapeeth University
Pune,India

***Abstract*— A peer-to-peer, commonly abbreviated to P2P, is any distributed network architecture composed of participants that make a portion of their resources (such as processing power, disk storage or network bandwidth) directly available to other network participants, without the need for central coordination instances (such as servers or stable hosts). Peers are both suppliers and consumers of resources, in contrast to the traditional client–server model where only servers supply, and clients consume. Peer-to-peer was popularized by file sharing systems like Napster During the last few years the distributed hash table (DHT) has emerged as a flexible and general architecture that can support a large variety of applications including file sharing, storage systems, query processing, name services and communication services. A DHT manages a global identifier (ID) space that is partitioned among n nodes organized in an overlay network. To partition the space, each node is given a unique ID x and owns the set of IDS that are "closest" to x. Each object is given an ID and the DHT stores an object at the node which owns the object's ID**

***Keywords*— Load balance, structured peer to peer system, distributed hash table.**

I. Introduction

Structured peer-to-peer (P2P) overlay networks like Distributed Hash Tables (DHTs) map data items to the network based on a consistent hashing function. Such mapping for data distribution has an inherent load balance problem. Thus, a load balancing mechanism is an indispensable part of a structured P2P overlay network for high performance. The rapid development of P2P systems has posed challenges in load balancing due to their features characterized by large scale, heterogeneity, dynamism, and proximity. An efficient load balancing method should flexible and resilient enough to deal with these characteristics. Structured P2P systems based on the DHT mechanism have proven to be an effective design for resource sharing on a global scale and on top of which many applications have been designed such as file sharing, distributed file systems, real-time streaming, and distributed processing. In these systems, each data item is mapped to a unique identifier ID drawn from an identifier space. The identifier space is partitioned among the nodes so that each node is responsible for a portion of the ID space, called zone, and storing all the objects that are mapped into its zone.

One central challenge in the DHT design is how to balance the load across the nodes in the system. Most P2P systems that provide a DHT abstraction distribute objects among "peer nodes" by choosing random identifiers for the objects. In the case of a homogeneous system where all nodes have the same capacity, DHTs can exhibit an O(1og n) imbalance factor. Besides, P2P systems can be highly heterogeneous, i.e. they may consist of peers that range from old desktops behind modem lines to powerful servers connected to the Internet through high-bandwidth lines. The imbalance can significantly increase as the heterogeneity of the system increases.

Two classes of solutions have been proposed so far to address this challenge. Solutions in the first class use the concept of Virtual Servers [VS]. Each physical node instantiates with one or more virtual servers with random ID's that act as peers in the DHT. In the case of a homogeneous system, maintaining O(1og n) virtual servers per physical node reduces the load imbalance to a constant factor. To handle heterogeneity, each node picks a number of virtual servers proportional to its capacity. Unfortunately, virtual servers incur a significant cost: a node with k virtual servers must maintain k sets of overlay links. Typically k = O(log n.), which leads to an asymptotic increase in overhead.

The second class of solutions uses just a single ID per node. However, all such solutions must reassign IDS to maintain the load balance as nodes arrive and depart the system. This can result in a high overhead because it involves transferring objects and updating overlay links. In addition, none of these solutions handles heterogeneity directly, although they could be combined with the virtual server technique. Also, for development of next-generation internet infrastructure, application layer peer-to-peer (P2P) networks are considered to be important along with the load balancing policies. In [1], the problem of load balancing in such P2P systems is addressed. The space of designing load-balancing algorithms is explored such that uses the notion of "virtual servers". Also, the iterative algorithmic approach for space exploration of solution is considered to find a best reassignment of servers for load balancing.

II. FORMALIZATION OF THE PROBLEM

According to studies in previous work on consistent hashing in [4] and [5], it assumed that nodes were aware of most other nodes in the system, making it impractical to scale to large number of nodes. In contrast, each Chord node needs "routing" information about only a few other nodes. Because the routing table is distributed, a node resolves the hash function by communicating with a few other nodes. In the steady state, in an N-node system, each node maintains information only about O(logN) other nodes, and resolves all lookups via O(logN) messages to other nodes. Chord maintains its routing information as nodes join and leave the system; with high probability each such event results in no more than $O(\log^2 N)$ messages. Consistent hashing assigns keys to nodes as follows. Identifiers are ordered in an identifier circle modulo 2^m. Key is assigned to the first node whose identifier is equal to or follows k in the identifier space. This node is called the successor node of key k, denoted by successor(k). If identifiers are represented as a circle of numbers from 0 to 2^m-1 , then successor(k) is the first node clockwise from k. Figure 2.1 shows an identifier circle with m=3. The circle has three nodes: 0, 1, and 3. Successor of identifier 1 is node 1, so key 1 would be located at node 1. Similarly, key 2 would be located at node 3, and key 6 at node 0. Consistent hashing is designed to let nodes enter and leave the network with minimal disruption. To maintain the consistent hashing mapping when a node n joins the network, certain keys previously assigned to n's successor now become assigned to n. When node n leaves the network, all of its assigned keys are reassigned to n's successor. No other changes in assignment of keys to nodes need occur. In the example above, if a node were to join with identifier 7, it would capture the key with identifier 6 from the node with identifier 0.

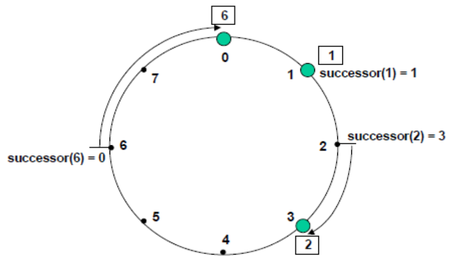

Fig. 1 An identifier circle consisting of the three nodes 0, 1,and 3.

In this example, key 1 is located at node 1, key 2 at node3, and key 6 at node 0

THEOREM: For any set of N nodes and K keys, with high probability:

1. Each node is responsible for at most (1+ϵ)K/N Keys.
2. When an (N+1)st node joins or leaves the network, responsibility for O(K/N) keys changes hands.

When consistent hashing is implemented as described, the theorem proves a bound of ϵ= O(logN). The consistent hashing shows that ϵ can be reduced to an arbitrarily small constant by having each node run O(logN) "virtual nodes" each with its own identifier.

Chord simplifies the design of peer-to-peer systems and applications based on it by addressing difficult problems as Load balance, Decentralization, Scalability, Availability, Flexible naming. In the implementation, the Chord software takes the form of a library to be linked with the client and server applications that use it. The application interacts with Chord in two main ways. First, Chord provides a lookup (key) algorithm that yields the IP address of the node responsible for the key. Second, the Chord software on each node notifies the application of changes in the set of keys that the node is responsible for. This allows the application software to, for example, move corresponding values to their new homes when a new node joins.

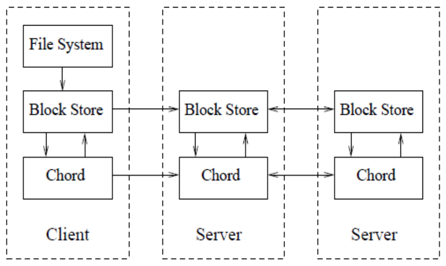

Fig. 2. Structure of an example Chord-based distributed storage system

Figure 2 shows a possible three-layered software structure for a cooperative mirror system. The highest layer would provide a file like interface to users, including user-friendly naming and authentication. This "file system" layer might implement named directories and files, mapping operations on them to lower-level block operations. The next layer, a "block storage" layer, would implement the block operations. It would take care of storage, caching, and replication of blocks. The block storage layer would use Chord to identify the node responsible for storing a block, and then talk to the block storage server on that node to read or write the block.

At its heart, Chord provides fast distributed computation of a hash function mapping keys to nodes responsible for them. It uses consistent hashing which has several good properties. With high probability the hash function balances load (all nodes receive roughly the same number of keys). Also with high probability, when an Nth node joins (or leaves) the network, only an O(1/N) fraction of the keys are moved to a different location, this is clearly the minimum necessary to maintain a balanced load.

III. PROPOSED WORK

Distributed hash tables (DHTs) are a class of decentralized distributed systems that provide a lookup service similar to a hash table; (key, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows DHTs to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

DHTs form an infrastructure that can be used to build more complex services, such as distributed file systems, peer-to-peer file sharing and content distribution systems, cooperative web caching, multicast, anycast, domain name services, and instant messaging.

DHTs characteristically emphasize the following properties:

- Decentralization: the nodes collectively form the system without any central coordination.
- Scalability: the system should function efficiently even with thousands or millions of nodes.
- Fault tolerance: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.

A key technique used to achieve these goals is that any one node needs to coordinate with only a few other nodes in the system – most commonly, O(log n) of the n participants – so that only a limited amount of work needs to be done for each change in membership. Some DHT designs seek to be secure against malicious participants and to allow participants to remain anonymous, though this is less common than in many other peer-to-peer (especially file sharing systems); see anonymous P2P. Finally, DHTs must deal with more traditional distributed systems issues such as load balancing, data integrity, and performance.

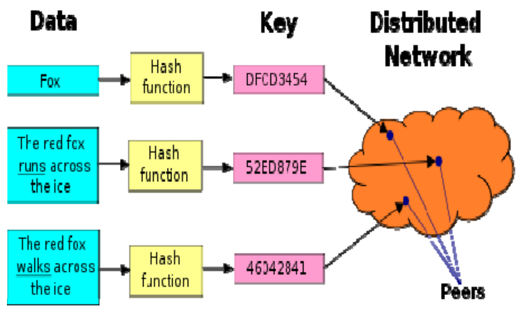

Fig. 3. Distributed Hash Tables

The structure of a DHT can be decomposed into several main components. The foundation is an abstract keyspace, such as the set of 160-bit strings. A keyspace partitioning scheme splits ownership of this keyspace among the participating nodes. An overlay network then connects the nodes, allowing them to find the owner of any given key in the keyspace. Once these components are in place, a typical use of the DHT for storage and retrieval might proceed as follows. The keyspace is the set of 160-bit strings. To store a file with given filename and data in the DHT, the SHA-1 hash of filename is generated, producing a 160-bit key k, and a message put (k,data) is sent to any node participating in the DHT. The message is forwarded from node to node through the overlay network until it reaches the single node responsible for key k as specified by the keyspace partitioning. That node then stores the key and the data. Any other client can then retrieve the contents of the file by again hashing filename to produce k and asking any DHT node to find the data associated with k with a message get(k). The message will again be routed through the overlay to the node responsible for k, which will reply with the stored data.

In [1] algorithm uses the concept of virtual servers. A virtual server presents a peer in the DHT i.e., the storage of data items and routing happen at the virtual server level rather than at the physical node level. A physical node hosts one or more virtual servers. Load balancing is achieved by moving virtual servers from heavily loaded physical nodes to lightly loaded physical nodes.

IV. PERFORMANCE EVALUATION

To evaluate the performance of the algorithm on all the instances, it is desirable to evaluate them when only feasible instances are considered. However, deciding if an instance has any feasible solutions is NP-hard, as mentioned before. There- fore, we define the following necessary conditions. We call the instances satisfying these conditions to be admissible instances. Clearly, instances that have feasible solutions must be admissible but not vice versa: The total workload should be smaller than or equal to the total capacity.

The maximum workload of the VSs should be smaller than or equal to the maximum capacity of the nodes, to ensure that at least one node capable of storing the largest VS exists.

The following are the main performance metrics used in this paper:

- Essential challenge in Peer-to-Peer systems is to locate data item in structured peer to peer system.
- Where the item shall be stored by the provider?
- How does a requester find the actual location of an item?
- Also the issues of scalability that is to keeping the complexity for communication and storage scalable, and Robustness and resilience in case of faults and frequent changes are to be tracked

Following figure shows management and retrieval of data

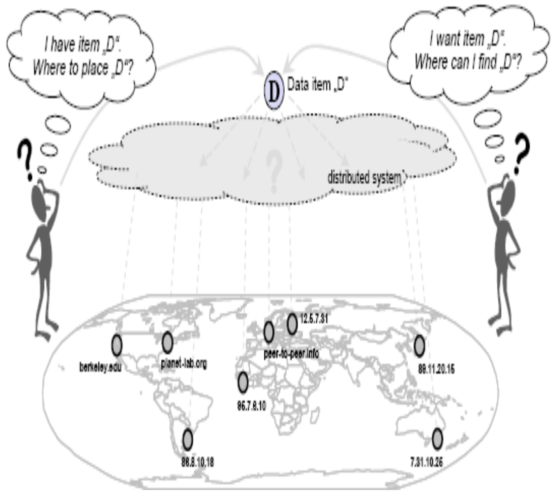

Fig. 4. Management and Retrieval of Data

As due to centralized indexing, there was a overhead of communication to the state of node as depicted in Fig. 4(a).There was need of Distributed Indexing structure which should be a intermediate scalable solution which lead to the implementation of DHT as in Fig. 4(b).

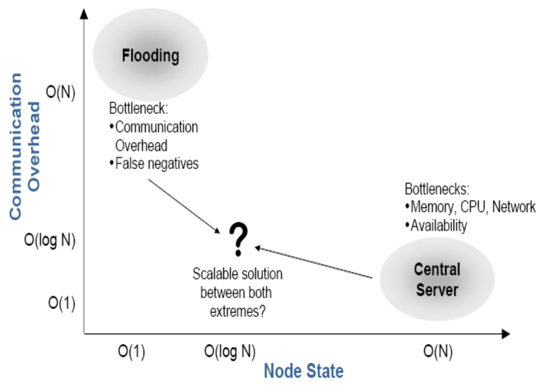

Fig. 4(a) Need of Distributed Indexing

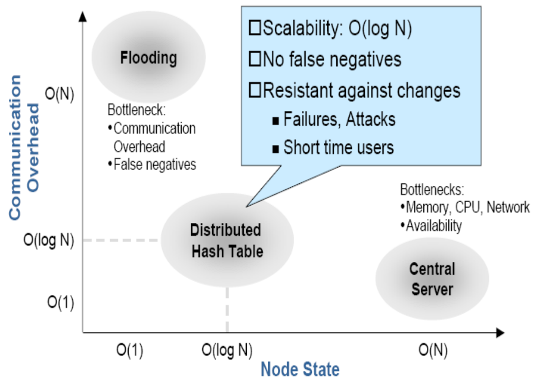

Fig. 4(b) Motivation to Distributed Indexing

V. CONCLUSION AND FUTURE WORK

The work have achieved the migration-based approach to load balancing in the term of virtual server (VS), which helps to under load the overloaded physical node by moving portions of the load dynamically. Several objectives of this implementation are achieved as Configuring Service/s, Load distribution using DHT technique for Server Reassignment. The effect of heterogeneous load on system implementation & cross platform usage is also analyzed. This work demonstrated the VS framework for solving the load balance problem in a structured P2P system.

We wish that our effort will provide a backbone structure for the further enhancements of the system. To investigate the following important issues in the future is needed: It is intended to explore other cost-reducing neighbourhoods to further improve the Dual Space Local Search algorithm. As the variance of a VS workload has a significant impact on the success ratio performance, the plan is to investigate VS merging and splitting strategies to enhance the performance of the algorithms. Also to perform a more in-depth study of issues in the dynamic scenario in which a node joins and leaves the system.

REFERENCES

1)Chyouhwa Chen, Kun-Cheng Tsai "The Server Reassignment Problem For Load Balancing In Structured P2P Systems ", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 19, NO. 2, FEBRUARY2008.
2) F. Dabek, M. Kaashoek, D. Karger, D. Morris, and I. Stoica, "Wide- Area Cooperative Storage with CFS," Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01), pp.202-215,Oct.2007.
3) X. Wang and D. Loguinov, "Load-Balancing Performance of Consistent Hashing: Asymptotic Analysis of Random Node Join," IEEE/ACM Trans. Networking, vol. 15, no. 5, Oct.2007.
4) 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 '03), Feb.2006.
5) P.B. Godfrey and I. Stoica, "Heterogeneity and Load Balance in Distributed Hash Tables," Proc. IEEE INFOCOM,2005.
6) B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems,"Proc.IEEE.INFOCOM,2006.
7) P.B. Godfrey and I. Stoica, "Heterogeneity and Load Balance in Distributed Hash Tables," Proc.IEEE INFO 2005.