Survey of Load Balancing Approachs in Peer-To-Peer Network

Vishakha Patange, D.D.Gatade

Abstract*— In recent years, structured peer-to-peer (P2P) has gained an important role in the design of large-scale distributed systems. However, due to their strict data placement rules, they are often prone to three main load imbalances, i.e., range, data, and execution skew. Further imbalance may result due to non-uniform distribution of objects in the identifier space and a high degree of heterogeneity in object loads and node capacities. A node's load may vary greatly over time since the system can be expected to experience continuous insertions and deletions of objects, skewed object arrival patterns, and continuous arrival and departure of nodes. A virtual server looks like a single peer to the underlying DHT, but each physical node can be responsible for more than one virtual server. Load balancing among application layer peer-to-peer (P2P) networks is critical for its effectiveness but, is considered to be the most important development for next-generation internet infrastructure. Most structured P2P systems rely on ID-space partitioning schemes to solve the load imbalance problem. P2P system harnesses the resources of large populations - networked computers in a cost-effective manner such as the storage, bandwidth, and computing power. In structured P2P systems, data items are spread across distributed computers (nodes), and the location of each item is determined in a decentralized manner.

***Index Terms*—About four key words or phrases in alphabetical order, separated by commas.**

I. INTRODUCTION

PEER-TO-PEER systems have emerged as an appealing solution for sharing and locating resources over the Internet. Each object (data item) that enters the system has an associated load, which might represent, for example, the number of bits required to store the object, popularity of object, or the amount of processer time needed to serve the object. Each objects also has a movement cost, which we are charged each time we move object between nodes. An assumption is made that the cost is same regardless of which two nodes are involved in the transfer. An object's load may or may not be related to its movement cost. The basic approach to the load balancing issue in structured P2P networks or distributed hash tables (DHTs) is consistent hashing.

There are two main goals to be achieved, minimize the load balance and minimize the amount of load moved. If the hot peers become bottleneck, it leads to increased user response time and significant performance degradation of the system. Hence the load balancing mechanism is necessary in such cases.

Manuscript received on May, 2013.

Vishakha Patange, M.E. (Computer Networks), Pune University, S.C.O.E. Pune., India.

D. D. Gatade, M.E. (Computer Networks), Pune University, S.C.O.E. Pune., India.

With the notion of virtual servers, peers participating in a heterogeneous, structured peer-to-peer (P2P) network may host different numbers of virtual servers, and by migrating virtual servers, peers can balance their loads proportional to their capacities. The security vulnerabilities are analyzed of the typical DHT load balancing mechanism; then propose an algorithm that both facilitates good performance and does not dilute security.

II. LITERATURE REVIEW

A. ID Management Algorithm:

The ID management algorithm presented here is a greedy distributed algorithm that directs joining peers to highly-frequented regions of the ID space. It is based on the idea that peers responsible for these regions are most likely to be overloaded [1]. To identify these highly-frequented regions this algorithm collects the statistics on the utilization of the peers' overlay links during the regular operation of the P2P network, i.e., without generating additional messaging overhead.

The type of balancing technique used here is Data and Execution balancing. This algorithm collects statistics of overlay link usage during normal operation and uses this information to provide suitable IDs to joining peers [1]. Without using regular maintenance messages, it improves the rate of successfully answered requests. The ID Management directs joining peers to highly-frequented regions of the ID space [1]. It is based on the idea that peers responsible for these regions are most likely to be overloaded. To identify these highly-frequented regions this algorithm collects the statistics on the utilization of the peers' overlay links during the regular operation of the P2P network, i.e., without generating additional messaging overhead [1].

This algorithm mainly presents the method ROUTE, which is in charge of delivering an incoming *msg* if the current peer is responsible for the destination ID *destID* or to forward it to closer to its destination [1]. Each peer consists of the routing table (RT) with m entries which maintains the count, *ip* address and ID. Each peer calculates its Join Link Table (JT) which contains all routing table entries. The normalization process used for the implementation of the ID management algorithm is based on network Chord [1]. Besides, RT and JT, each peer maintains the third table, called as Destination Table (DT). It keeps track of the peer's workload W. Each time the responsibility range of peer changes, because an adjacent peer is joining or leaving, the table is reset [1].

B. Intra-Cluster and Inter-Cluster Load Balancing

The concept of clustering is basically the connecting of two or more nodes in such a way that they behave like a single node. Clustering is used for parallel processing, load balancing and

fault tolerance. Each cluster has a unique ID. The nodes are grouped into strong and weak clusters based on their weight vector which comprises of the parameters like: Available Capacity, CPU Speed, Memory Size and Access Latency.
In Intra-Cluster Load Balancing Algorithm, the cluster leader receives the information periodically regarding the loads and available disk space of the peers. Based on load the cluster leader creates a sorted list of the peers such that the first element of the list is the heavily loaded peer [5]. It is monitored that, only for a particular periodic time intervals cluster leader checks for the load imbalance and not whenever any peer joins/leaves the system. Custer leader corrects the load imbalances which are caused by some peers while joining/leaving the system [5]. The cluster leader checks whether its load exceeds the average load of its neighboring cluster leaders by more than 10% of the average load. If it exceeds, then it determines the hot data items which should be moved. Thus load is balanced.
In case of intra-cluster load balancing, some of the decisions are critical to system performances regarding when to trigger the load balancing mechanism, hotspot detection and the amount of data to be replicated. The cluster leader receives the information periodically regarding the loads and available disk space of the peers. Based on load the cluster leader creates a sorted list of the peers such that the first element of the list is the heavily loaded peer [5]. The load balancing is achieved by replicating the hot data from the first peer in the list to the last peer and the second peer to the second last peer and so on. If the load difference between the peers exceeds a pre-specified threshold, then the data will be replicated.

C. Load Balancing Algorithm in Dynamic Structured P2P Systems using Directories

The type of balancing technique used here is Load balancing. The basic idea of load balancing algorithm is to store load information of the peer nodes in a number of directories which periodically schedule reassignments of virtual servers to achieve better balance [6].
Here, each directory has an ID known to all nodes and is stored at the node responsible for that ID. Each directory collects load and capacity information from nodes which contact it. When node's utilization jumps above a parameterized threshold, it immediately reports to the directory which it has contacted recently [6]. It then schedules immediate transfer from present node to the lightly loaded nodes.

2.1 Proposed System

The load balancing scheme here is not restricted to a particular type of resource (e.g., storage, bandwidth, or CPU). However, two assumptions are considered in our work. First, assume that there is only one bottleneck resource in the system, leaving multi-resource balancing to our future work. Second, assume that the load on a virtual server is stable over the timescale it takes for the load balancing algorithm to perform. Basically, the load balancing scheme consists of four phases:
a. Load balancing information (LBI) aggregation. Aggregate load and capacity information in the whole system.

b. Node classification. Classify nodes into overloaded (heavy) nodes, under loaded (light) nodes, or neutral nodes according to their loads and capacities.

c. Virtual server assignment (VSA). Determine virtual server assignment from heavy nodes to light nodes in order to have heavy nodes become light. The VSA process is a critical phase because it is in this phase that the proximity information is used to make our load balancing scheme proximity-aware.
d. Virtual server transferring (VST). Transfer assigned virtual servers from heavy nodes to light nodes. We allow VSA and VST to partly overlap for fast load balancing.

Also, few attentions have been paid on security threats introduced by the load balancing. An SLBA algorithm is implemented for security issues. SLBA works as follows: (1) at joining time, based on targeted interval verifiable ID generation, an unique semi-CA server generates a set of verifiable IDs for node, which can limit any fundamental decrease in security and greedily reduces discrepancies between capacity and load; (2) during its run-time, experiencing overload node should execute security-aware load transfer algorithm, which can significantly raise the convergence rate and load transfer security. The advantages are :

- Better load balancing than existing schemes.
- Peers can compute their expected loads and reallocate their loads in parallel.

III. LOAD BALANCING MODULEWISE DISTRIBUTION

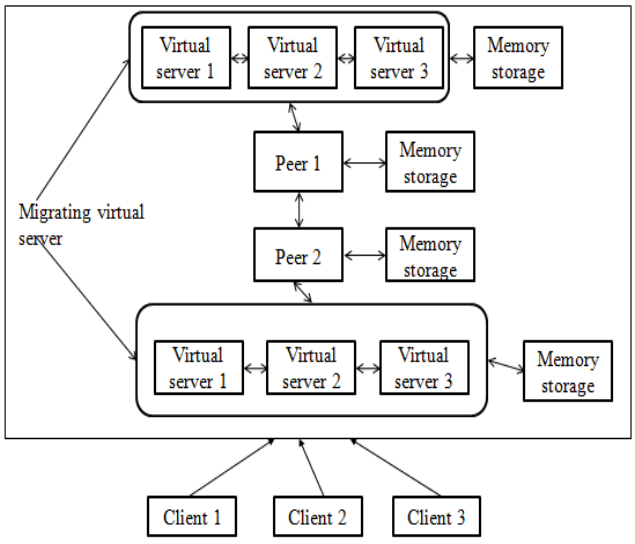

Fig.1 System Architecture

Module 1: Network Creation

In the module designing of windows for the peer page is performed. These windows are used to send a message from one peer to another. Omnet++ is used for the simulation in network creation. Swing package is used to design the User Interface. Swing is widget toolkit, part of an API for providing a graphical user interface (GUI) for the coding. In this module main focus is on the login design page with the Partial knowledge information. Every user should know about their neighbor details so that they can share the information in the network. In this situation main focus is the DHT table in the network. Distribute Hash Table maintain a neighbor information in order to get the partial information about the clients. Like Client Ip Address, Port No. etc. With this

information client can connect and communicate with other clients in the network and share their data.
Input: Design the page and connecting to each other.
Output: Get the network according to the need.
Module 2: Implementation of the client
In this module client home page design is developed. Here, first the client is going to login. While entering into a client home page client should connect with super peer for that Super Peer Ip and Port No. of Super Peer which is already running in network should be provided. Super Peer will accept the request from the client and send the acknowledgement to the client. Now client can communicate with super peer. Client can browse the file from the system and upload the data to the super peer.
Input: Enter client port and connect with super peer.
Output: Client connected with super peer.

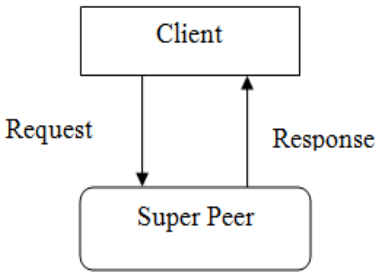

Fig.2 Client and Super Peer Communication

Module 3: Determination of the peer
In this module, design of super peer home page is developed. In this, super peer should connect with the virtual server so that it asks the number of virtual servers needed in this network. Then specify the number of virtual sever. So that super peer will connect with the virtual server. Super peer will have some capacity so it will check the memory first and if memory is available it will get the data from the client and store it in the memory if memory is not available in the Super peer it will check is there any virtual server so that it will upload the data to the virtual server.

Input: Enter the Super Peer Port No. and the N no. of Virtual Server.
Output: Super Peer created.
Module 4: Establishment of the Virtual Server
Here, Virtual server is developed. Number of virtual servers can be run simultaneously using simulation manner. The virtual server will handle the request and response to the super peer according to its capacity. Super Peer can have N no. of virtual server as its need. Virtual Server will have some capacity. If super peer capacity is over it will store the data into virtual server. If virtual server capacity is no more then it will response to the super peer to request another super peer to migrate some of its virtual server.
Input: Enter the Virtual Server port no. and run it.
Output: Virtual Server created.
Module 5: Migration of the Work Load
The load balancing in the peer to peer network is accomplished in this module, it manages the work load in the network when the excess load occurs in the network and the node is heavily-loaded the loads are transferred between virtual servers. If the peer node has no sufficient virtual server for balancing the load then the peer request for the other super peer in the network and asks for the virtual server and then balances its load.
Input: Request for the virtual server in the other super peer node.
Output: Excess load is transferred to the other virtual server through migration.

IV. CONCLUSION

A load balancing algorithm for DHTs with virtual severs is studied which represents the system state. With the DHT used, each peer identifies whether it is under loaded and then reallocates its loads if it is overloaded. In particular, while the centralized directory and tree-based approaches introduce hotspots to the system, the participating peers perceive nearly identical workloads in manipulating the load balancing algorithm.

REFERENCES

[1] Daniel Warneke, Christian Dannewitz, "Load Balancing in P2P Networks: Using Statistics to Fight Data and Execution Skew," IEEE Trans., Oct. 2009.
[2] Hung-Chang Hsiao, Member, IEEE Computer Society, Hao Liao, Ssu-Ta Chen, and Kuo-Chan Huang, "Load Balance with Imperfect Information in Structured Peer-To-Peer Systems," IEEE Trans. Parallel and Distributed Systems, vol. 22, no. 4, Apr. 2011.
[3] Leonidas Lymberopoulos, Symeon Papavassiliou, Vasilis Maglaris, "A Novel Load Balancing Mechanism for P2P Networking," ICST, October 2007.
[4] Quang Hieu Vu, Member, IEEE, Beng Chin Ooi, Martin Rinard, and Kian-Lee Tan, " Histogram-Based Global Load Balancing in Structured Peer-To-Peer Systems," IEEE Trans. On Knowledge and Data Engineering Vol.21, No.4, April 2009.
[5] S. Ayyasamy, S. N. Sivanandam, " A Cluster Based Replication Architecture for Load Balancing in Peer-To-Peer Content Distribution, "International Journal of Computer Networks and Communications (IJCNC) Vol.2, No.5, September 2010.
[6] 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.
[7] S. S. Patil, S.K. Shirgave, " Load Balancing in Structured P2P Systems using Server Reassignment Technique," International Journal of Computer Applications (0975-8887) Volume1- No.4.