LOAD BALANCING WITH PARTIAL KNOWLEDGE OF SYSTEM IN PEER TO PEER NETWORKS

R. Vijayalakshmi and S. Muthu Kumarasamy
Dept. of Computer Science & Engineering, S.A. Engineering College
Anna University, Chennai, India.

ABSTRACT

Load balancing is a critical issue for the efficient operation of peer-to- peer networks. The goal of P2P systems is to harness all available resources (storage, bandwidth, and CPU) in the P2P network so that users can access all available objects efficiently.P2P aims to directly balance the distribution of items among the nodes. 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. Peers participating in a Distributed Hash Table (DHT) are often heterogeneous. Potential P2P substrates are based on Distributed Hash Tables. The existing and decentralized load balance algorithms designed for the heterogeneous, structured P2P networks either explicitly construct auxiliary networks to manipulate global information or implicitly demand the P2P substrates organized in a hierarchical fashion. Without relying on any auxiliary networks and independent of the geometry of the P2P substrates, we present, in this paper, a novel efficient, proximity-aware load balancing algorithm by using the concept of virtual servers, that is unique in that each participating peer is based on the partial knowledge of the system to estimate the probability distributions of the capacities of peers and the loads of virtual servers, resulting in imperfect knowledge of the system state. With the imperfect system state, peers can compute their expected loads and reallocate their loads in parallel.

KEYWORDS*-Peer-to-Peer Systems, Load Balance, Heterogeneity, Decentralized.*

I. INTRODUCTION

P2P computing is the sharing of computer resources and services by direct exchange between systems. Second generation P2P overlay networks are self-organizing, load balanced, fault-tolerant, and scalable guarantees on numbers of hops to answer a query. PEER-TO-PEER (P2P) systems make it possible to harness resources such as the storage, bandwidth, and computing power of large populations of networked computers in a cost-effective manner.

In this paper we focus on the virtual server (VS) or migration-based approach to load balancing in this paper, which can move the portions of the load off an overloaded physical node dynamically and has already been explored in a number of studies. As peers participating in a DHT are often heterogeneous, the work in [2] introduces the notion of virtual servers to cope with the heterogeneity of peers. Participating peers in a DHT can host different numbers of virtual servers, thus taking advantage of peer heterogeneity

Our algorithm uses the concept of *virtual servers* proposed [3].A virtual server represents a peer in the DHT; that is, 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 physical nodes to lightly loaded physical nodes.

The DHT is based on the Chord protocol. The *Chord protocol* supports just one operation: given a key, it maps the key onto a node. Depending on the application using Chord, that node might be responsible for storing a value associated with the key. Chord uses consistent hashing to assign keys to Chord nodes.

II. OUR PROPOSAL

In this paper we make the following contributions:

- We propose an algorithm which to the best of our knowledge is the first to provide dynamic load balancing in heterogeneous, structured P2P systems.
- We study the proposed algorithm by using extensive simulations over a wide set of system scenarios and algorithm parameters. We are solving the problem of load balancing in a completely decentralized manner.

A load balancing algorithm should strive to achieve the following (often conflicting) goals:

- Minimize the load imbalance. To provide the best quality of service, every node would have the same utilization. Furthermore, for resources with a well-defined cliff in the load-response curve, it is of primary importance that no node's load is above the load at which the cliff occurs. We can take this point to be the capacity of the node.
- Minimize the amount of load moved. Moving a large amount of load uses bandwidth and may be infeasible if a node's load changes quickly in relation to the time needed to move objects.

A. Algorithm Sketch

In this paper, we assume that the entire hash space provided by a DHT and each virtual server in the DHT has a unique ID selected independently and uniformly at random from the space. Let N be the set of participating peers, and V be the set of virtual servers hosted by the peers in N in the DHT. Denote the set of virtual servers in peer i by Vi. Each peer in our proposal estimates the load, which is denoted by Ti, that it should perceive, where A is an estimation for the expected load per unit capacity.

If the current total load of i is greater than Ti (i.e., i is overloaded), then i migrates some of its virtual servers to other peers. Otherwise, i is underloaded, which does nothing but waits to receive the migrated virtual servers. For an overloaded peer (e.g., peer i), i picks those virtual servers for migration, such that 1) i becomes underloaded, and 2) the total movement cost, MC, in (3) is minimized due to the reallocation. If i is an underloaded peer, then i may be requested to receive a migrated virtual server, and i accepts such a virtual server if the added load due to the virtual server will not overload itself; otherwise, i rejects such virtual server. Algorithm 1 (REALLOCATION), which given as follows illustrates our idea.

III. SYSTEM ARCHITECTURE

Algorithm 1(REALLOCATION), the challenges of implementing the algorithm are 1) how a peer precisely and timely estimates A and 2) how an overloaded peer seeks the peers to receive its migrated virtual servers for balancing the loads among peers. To deal with these issues, our idea is to represent the capacities of participating peers and the loads of virtual servers as the probability distributions, which are denoted by $Pr(X < x)$ and $Pr(Y < y)$, respectively. Both $Pr(X < x)$ and $Pr(Y < y)$ provide valuable information to help the participating peers estimate A, and the overloaded peers discover the underloaded peers to share their excess loads.

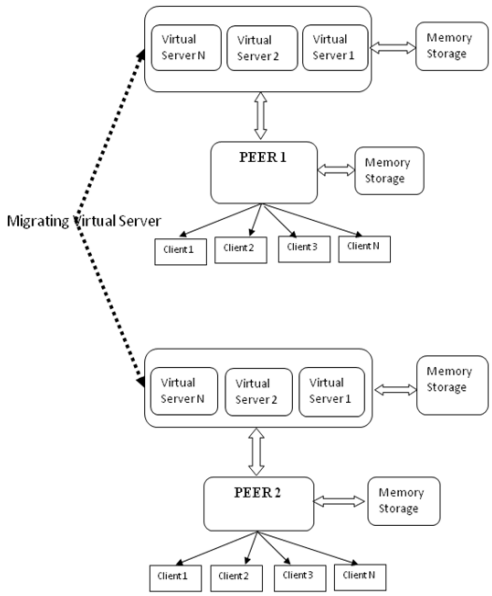

Figure 1 : System Architecture

IV. SYSTEM DESIGN

A. Background: Virtual Servers

The concept of virtual servers was first proposed in Chord to improve load balance. Like a physical peer node, a virtual server is responsible for a contiguous portion of the DHT's identifier space. A physical peer node can host multiple virtual servers and, therefore, can own multiple non-contiguous portions of the DHT's identifier space. Each virtual server participates in the DHT as a single entity.

B. System Overview

The load balancing scheme we present in this paper is not restricted to a particular type of resource (e.g., storage, bandwidth, or CPU). However, we make two assumptions in our work. First, we assume that there is only one bottleneck resource in the system, leaving multi-resource balancing to our future work. Second, we assume that the load on a virtual server is stable over the timescale it takes for the load balancing algorithm to perform. Basically, our load balancing scheme consists of four phases:

1. Load balancing information (LBI) aggregation. Aggregate load and capacity information in the whole system.
2. Node classification. Classify nodes into overloaded (heavy) nodes, underloaded (light) nodes, or neutral nodes according to their loads and capacities.
3. 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.
4. 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.

C. SYSTEM MODEL with the DHT

Chord simplifies the design of peer-to-peer systems and applications based on it by addressing these difficult problems

- Load Balance: Chord acts as a distributed hash function, spreading keys evenly over the nodes; this provides a degree of natural load balance.
- Decentralization: Chord is fully distributed; no node is more important than any other. This improves robustness and makes Chord appropriate for loosely organized peer-to-peer applications.

- Scalability: The cost of a Chord lookup grows as the log of the number of nodes, so even very large systems are parameter tuning is required to achieve this scaling.
- Availability: Chord automatically adjusts its internal tables to reflect newly joined nodes as well as node failures, ensuring that, barring major failures in the underlying network, the node responsible for a key can always be found.
- Flexible Naming: Chord places no constraints on the structure of the keys it looks up; the Chord keyspace is flat. This gives applications a large amount of flexibility in how they map their own names to Chord keys.

The application using Chord is responsible for providing any desired authentication, caching, replication, and user-friendly naming of data. Chord's flat key space eases the implementation of these features.

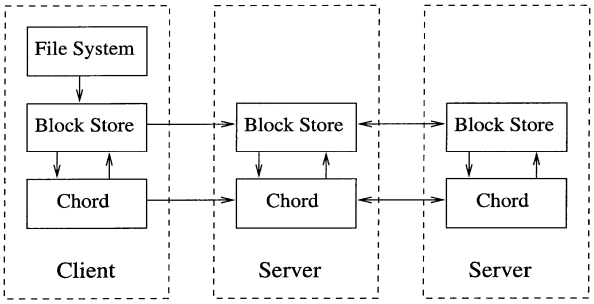

Figure 2 : Structure of an Example Chord-based Distributed Storage System

V. EXPERIMENTAL RESULTS

The results of the load balancing in the peer to peer networks and by migrating virtual servers the peer could balance the load proportional to their capacity, the load balancing is carried out by 4 modules in the paper. They are

- Network Creation
- Implementation of the Client
- Determination of the Peer
- Establishment of the Virtual Server
- Migration of the Work Load

NETWORK CREATION

In the module we design the windows for the peer page. 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 Sun Microsystems' Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs. In this module mainly we are focusing the login design page with the Partial knowledge information. Every user should know about their neighbor details so that they can share their information in the network. In this situation we mainly focus the DHT table in our network. Distribute Hash Table maintain a neighbor information so we can get the partial information about the clients. Like Client name, Ip Address, Port No etc. With this information client can 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 our need.

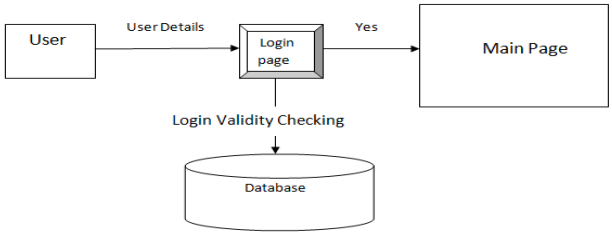

Figure 3: Creation of new users

IMPLEMENTATION OF THE CLIENT

In this module we develop a client home page design. this first the client is going to login. While entering into a client home page client should connect with super peer for that we have to provide a Super Peer Ip & Port No of Super Peer which is running already in the network. 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.

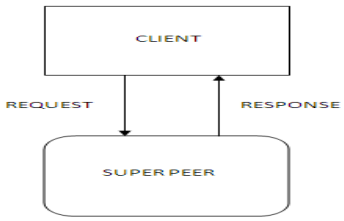

Figure 4: Client and super peer communication

Figure 5: Client Login Page

Figure 6: Client Uploading Files in the peer

DETERMINATION OF THE PEER

In this module we develop a super peer home page design. In this super peer should connect with the virtual server so that it asks the number of virtual server needed in this network. Then specify the number of virtual server. 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 is there so that it will upload the data to the virtual server.

Input: Enter the Super Peer Port No and get the N no of virtual server

Output: Super Peer created.

Figure 7: Super Peer Page

ESTABLISHMENT OF THE VIRTUAL SERVER

In this module develop a virtual server in java swing concept. You have to run as many virtual servers u need in the 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 it's the virtual server.

Input: Enter the Virtual Server port no and run it.

Output: Virtual server created.

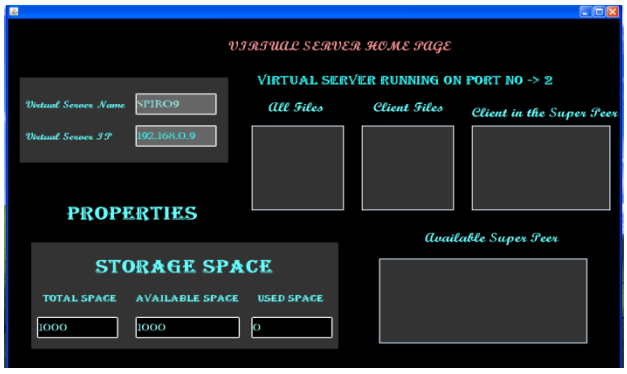

Figure 8: Virtual Server Home Page

MIGRATION OF THE WORK LOAD

The load balancing in the 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.

VI. SUMMARY

In this paper, we have presenting a novel load balancing algorithm for DHTs with virtual servers. Our proposal is unique in that we represent the system state with probability distributions. Unlike prior solutions that often rely on global knowledge of the system, each peer in our proposal independently estimates the probability distributions for the capacities of participating peers and the loads of virtual servers based on partial knowledge of the system. With the approximated probability distributions, each peer identifies whether it is under-loaded and then reallocates its loads if it is overloaded.

REFERENCES

[1] Hung-Chang Hsiao, HaoLiao,Ssu-ta Chen and Kuo-Chan Huang ,"Load Balance with imperfect information in Structured Peer-to-Peer Systems" April 2011
[2] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. aashoek, .Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," Feb. 2003.
[3] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," Nov. 2001.
[4] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I.Stoica, "Load Balancing in Dynamic Structured P2P Systems,", Mar. 2006.
[5] Y. Zhu and Y. Hu, "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems," Apr. 2005.
[6] H. Shen and C.-Z.Xu, "Locality-Aware and Churn-Resilient Load Balancing Algorithms n Structured P2P Networks," June 2007.
[7] L.M. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," May 1985.
[8] L.M. Ni, C.-W. Xu, and T.B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," Oct. 1985.
[9] C. Chen and K.-C. Tsai, "The Server Reassignment Problem for Load Balancing in Structured P2P Systems," Feb. 2008.
[10] T.L. Casavant and J.G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," Feb. 1988.
[11] Y. Zhu, "Load Balancing in Structured P2P Networks," July 2009.
[12] D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," June 2004.\
[13] W.K. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," Apr. 1970.
[14] S.M. Ross, "Markov Chains," Introduction to Probability Models, 2007.

BIOGRAPHY

R. Vijayalakshmi received the B.Tech degree in Information Technology from Anna University, Chennai, India in the year 2010, and she is a final year PG Student, M.E Computer Science and Engineering. Her areas of interest include Peer to Peer Computing, Mobile Computing.

S. MuthuKumarasamy received the BE degree in Computer Science and Engineering from Anna University Chennai-India in the year 2005 and ME degree in Computer Science and Engineering from Anna University Chennai-India in the year 2011.He is currently working as a lecturer in S.A Engineering College. His research interests include Dynamic load balancing algorithm design and analysis.