Study on Load Management for Hierarchical Peer-to-Peer File Search
（階層型ピア・ツー・ピアファイル検索のための負荷管理の研究）

by

CAO QI

July, 2014

A dissertation submitted to the Faculty of the Information Engineering Course of the Graduate School of Engineering of Hiroshima University in partial fulfillment of the requirements for the degree of Doctor of Engineering.

To all the people

who have helped me reach this far.

Acknowledgments

This thesis is the outcome of help and guidance of a lot of people. First and foremost, I am heartily thankful to my supervisor Prof. Satoshi Fujita, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. His serious attitude and boundless enthusiasm for research was an excellent example for me.

I would like to thank Assoc. Prof. Sayaka Kamei and Assist. Prof. Satoshi Taoka in Distributed Systems Laboratory, for their valuable comments during my research.

I would like to thank Mr. Masayuki Tamura and all staffs of Young Researchers Education Center, for their warm-hearted encouragement and support.

I would like to convey my special thanks to all colleagues, tutors and leaders when I interned at Sharp Corporation (2010), Lenovo (Japan) Ltd. (2011) and Core Corporation (2012). They shared a number of valuable insights about the electronic industry, and made me think "true meaning" of research from an industrial perspective.

I would like to convey my special thanks to Mr. Kazuyuki Nakagawa, and other alumni of Hiroshima University who are working at Renesas Electronics Corporation, for their valuable suggestions, discussions and comments during my job hunting activities. With their help, i can devote myself entirely to do research.

I would like to thank Bahaa Alghazawy, and all my friends and colleagues in Hiroshima University, for their kind encouragements and supports in life and research.

I express my sincere gratitude to the faculty members and staffs of the department of information engineering.

I am thankful to my parent for their constant support. I am equally thankful to my wife, for being patient, supportive and helping during preparation of this thesis.

Abstract

In a Peer-to-Peer (P2P) system, multiple interconnected **peers** or **nodes** contribute a portion of their resources (e.g., files, disk storage, network bandwidth) in order to inexpensively handle tasks that would normally require powerful servers. Since the emergency of P2P file sharing, *load balancing* has been considered as a primary concern, as well as other issues such as autonomy, fault tolerance and security. In a process of file search, a heavily loaded peer may incur a long latency or failure in query forwarding or responding. If there are many such peers in a system, it may cause link congestion or path congestion, and consequently affect the performance of overall system. To avoid such situation, some of general techniques used in Web systems such as caching and paging are adopted into P2P systems. However, it is highly insufficient for load balancing since peers often exhibit high heterogeneity and dynamicity in P2P systems. To overcome such a difficulty, the use of *super-peers* is currently being the most promising approach in optimizing allocation of system load to peers, i.e., it allocates more system load to high capacity and stable super-peers by assigning task of index maintenance and retrieval to them.

In this thesis, we focused on two kinds of super-peer based *hierarchical architectures* of P2P systems, which are distinguished by the organization of super-peers. In each of them, we discussed system load allocation, and proposed novel load balancing algorithms for alleviating load imbalance of super-peers, aiming to decrease average and variation of query response time during index retrieval process.

More concretely, in this thesis, our contribution to *load management* solutions for hierarchical P2P file search are the following:

- In Qin's hierarchical architecture, indices of files held by the user peers in the bottom layer are stored at the super-peers in the middle layer, and the correlation of those two bottom layers is controlled by the central server(s) in the top layer using the notion of tags. In Qin's system, a heavily loaded super-peer can move excessive load to a lightly loaded super-peer by using the notion of task migration. However, such a task migration approach is not sufficient to balance the load of super-peers if the size of tasks is highly imbalanced. To overcome such an issue, in this thesis, we propose two task migration schemes for this architecture, aiming to ensure an even load distribution over the super-peers. The first scheme controls the load of each task in order to decrease the total cost of task migration. The second scheme directly balances the load over tasks by reordering the priority of tags used in the query forwarding step. The effectiveness of the proposed schemes are evaluated by simulation. The result of simulations indicates that all the schemes can work in coordinate, in alleviating the bottleneck situation of super-peers.

- In DHT-based super-peer architecture, indices of files held by the user peers in the lower layer are stored at the DHT connected super-peers in the upper layer. In DHT-based super-peer systems, the skewness of user's preference regarding keywords contained in multi-keyword query causes query load imbalance of super-peers that combines both routing and response load. Although index replication has a great potential for alleviating this problem, existing schemes did not explicitly address it or incurred high cost. To overcome such an issue, in this thesis, we propose an integrated solution that consists of three replication schemes to alleviate query load imbalance while minimizing the cost. The first

scheme is an active index replication in order to decrease routing load in the super-peer layer, and distribute response load of an index among super-peers that stored the replica. The second scheme is a proactive pointer replication that places location information of an index, for reducing maintenance cost between the index and its replicas. The third scheme is a passive index replication that guarantees the maximum query load of super-peers. The result of simulations indicates that the proposed schemes can help alleviating the query load imbalance of super-peers. Moreover, by comparison it was found that our schemes are more cost-effective on placing replicas than other approaches.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Load Balancing Problem

Load balancing has been widely used as one of fundamental technologies in improving the performance of parallel and distributed computing [19, 89, 91], Web server systems [13, 14, 100] and so on. A classic load balancing problem in parallel and distributed computing is to distribute tasks (i.e., chunks of work) among a set of processors as evenly as possible. More concretely, this problem can be stated as follows: Suppose we have a set of tasks T to assign them to a set of processors P, such that only one task can run on a processor at any time. Given the execution time of each task, how should we assign them to processors to minimize the final completion time?

Finding the exact answer to this question is not as simple as it looks. In fact, this was one of the original problems shown to be NP-complete in [27]. In developing simple and effective load balancing algorithms, *randomization* has been proven to be a promising approach. An easy randomized load balancing problem can be analyzed using Balls and Bins model [51, 63]. Balls and Bins model can be described as follows: Consider a sequential process of throwing n balls into n bins. At every step, each ball is thrown into a uniformly random bin, independent of other balls. Let the load of a bin be the number of balls after all balls have been thrown. It is well-known that with high probability[1], the maximum load of bins will be approximately $\frac{\ln n}{\ln \ln n}$ [32]. In the previous process, at every step we pick a bin independently and uniformly at random and throw the ball into that bin. Instead if every time we pick two bins independently and uniformly at random and throw the ball into the bin with fewer balls, the expected maximum load of bins will become $\frac{\ln \ln n}{\ln 2}$. More generally, if we allow t random choices for every ball, the maximum expected load of bins will be $\frac{\ln \ln n}{\ln t}$. The tradeoff between the amount of randomness used and the load achieved is optimized when $t = 2$. This phenomenon is called the *power of two choices* [51]. For further information of load balancing problem in the field of parallel and distributed computing, the interested reader is referred to literature [19, 89, 91].

In the field of Web server systems, Web performance perceived by end users is already increasingly dominated by server delays, especially when contacting busy servers [15]. The architecture solutions for scalable Web server systems can be classified into two categories, i.e., single server scale-out and multiple server scale-out. With a rapid growing of end users and complexity of Web services, improving the power of a single server from either hardware or software is highly insufficient to solve the Web scalability problem in a foreseeable future. Thus, in order to provide scalable Internet services, load balancing is often used for distributing request load over multiple Web servers (sometimes known as a server farm or server cluster). Multiple server scale-up can be again classified into local scale-out and global scale-out. Local scale-out is to deploy local cluster of Web servers [13, 14, 100], while global scale-out is to manage load between geographically distributed servers or data centers [22, 34, 47]. DNS round-robin and “load balancer” are two common algorithms for load balancing of multiple servers [87].

Unlike Web server systems, **peer-to-peer** (P2P) systems have emerged as an

[1]It means with probability at least $1 - \mathrm{O}(1/n)$, where n is the number of balls.

alternative way to provide scalable network services over the Internet [21, 3, 41]. A P2P system consists of a large number of computers called **peers** or **nodes**, which are connected with a logical network called P2P overlay to realize a direct communication between them. Thus far, a number of important network services have been realized under the P2P model, which include BitTorrent [11], Skype [79] and SopCast [80]. File sharing is a dominant P2P application on the Internet, allowing users to easily contribute, search and obtain contents. Examples of P2P file sharing are Napster [54], Gnutella [2] and Kazaa [45].

In many distributed systems, network performance of a peer is a function of its load [42, 99, 90]. Figure 1.1 illustrates the relation among load, throughput and response time inside a peer. When the load is light, throughput is linearly proportional to the load and response time is almost unchanged. After the load reaches the network capacity, throughput won't increase much with the load. Instead, packets will be queued and the response time will become longer in this period. The response time would be significantly increased if packets get discarded due to buffer overflow.

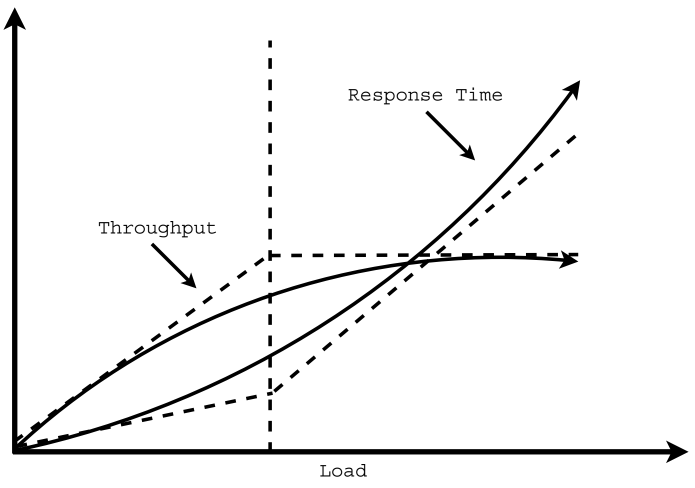


Figure 1.1: Network performance versus load

Thus, in P2P systems[2], a heavily loaded peer may incur a long latency or failure in forwarding or answering queries. If there are many such peers in a P2P system, it may incur link congestion or path congestion (i.e., a path contained one or more congested links) that affect the performance of overall system. To avoid such situation, some of general techniques used in Web systems such as cache and paging are adopted into P2P systems. However, it is highly insufficient for load balancing of peers since the characters of P2P systems such as heterogeneity and dynamicity of peers are ignored. Therefore, it is necessary to study load management from architectures and algorithms for P2P file search, in order to decrease average and variation of response time to queries.

Load management for P2P systems is a rather wide issue, we should make it clear with the following questions [74].

- What causes load problem? Most of the causes come from either an inappropriate architecture or the search algorithm used in the architecture. Once the load problem is recognized, load metric needs to be determined in order to evaluate "value" of the load.

- How to reduce the load? or How to balance it? In the former case, it needs to find ways to either reduce the load that is generated by the input of the system, or optimize the path length for the whole set of queries to reduce the routing load in P2P systems. In the latter case, it needs to identify available candidate peers to share the load.

- Which peer will be chosen as a candidate peer? Selection of candidate peers would affect the efficiency and effectiveness of load reduction and/or load balancing.

[2]We use "P2P systems" to denote "P2P file sharing systems" in the following.

We give an example to demonstrate the analysis of load problem and the possible solutions to it. Assume that a P2P overlay is used for file search. In the overlay, each peer owns some files. Consider a peer owns much more files compared to the other peers of the system. This peer needs more storage resource to allocate the files, and it potentially needs to serve more requests if popularity of files are not too disparate. The load problem is the uneven distribution of number of files at peers. On the other hand, if popularity of files is highly skewed, a peer that owns popular files would serve more requests than the other peers. In turn, the load problem would be the uneven distribution of number of queries received by peers. In the overlay, a peer can reduce its load by deleting some files it served, obviously it is not a cleaver solution since it highly reduce the availability of the overlay. Alternatively, if a highly loaded peer could move some files to other peers, such as neighbor peers, the load would be balanced in a feasible way.

Through the above example, we showed that common load problems of P2P systems, and presented possible solutions to the load problems. However, these solutions may be limited by the architecture itself, such as it may not allocate tasks to appropriate peers. Consequently, load management has to be thought of from the design time of systems, in order to avoid later load problems. In this thesis, we manage the load focusing on both of architectures and algorithms.

1.2 Background of Peer-to-Peer Systems

In P2P systems, index is a common and useful data structure for object (i.e., file) retrieval. Mainly an index is a set of entries associated to a given keyword, and an entry is a triple that contains keyword, object metadata and object location. Index retrieval is a key component in object retrieval, i.e., a peer issues a query that contains several keywords, and later the query is forwarded in the system until a destination peer is found or some failure occurs. Depending on the overlay, the process of index retrieval can be realized in different ways. Usually, a query tends to use shortest communication path, in order to obtain a short response time. For example, a centralized server can offer index retrieval service to a set of clients. The clients always communicate with the server to obtain the destination peer. As an alternative way, a set of peers can be used to relay the query to the destination peer. In this way, since task of index retrieval is distributed over all peers, it could be more scalable than the previous approach. But, the response time would become longer than the previous approach.

In the following sections of this chapter, we overview current P2P architectures and search algorithms, with a focus on the load management in the systems.

1.2.1 P2P Architectures

By their "degree of centralization", existing P2P architectures can be classified into three categories [3, 62]; i.e., centralized type, pure type and hierarchical type (hybrid type). Pure type and hierarchical type are also categorized into decentralized type. Details of each architecture is illustrated as follows:

- **Centralized type**

  Index of all files are uploaded to a centralized server, and maintained by it.

  Napster [54] is a centralized P2P system consisting of a number of user peers and an index server, where the server keeps the information on the location of shared mp3 files maintained by participating peers. Figure 1.2 illustrates the architecture of Napster. When the server receives a query from a user peer, it refers its index, returning a list of user peers that hold the file matching to the query. A requester peer can open a direct connection with the peer that holds the requested file, and download it (See Figure 1.2). This architecture provides efficient query processing and low response time to the queries under a low server usage. However, it is well known that it has serious drawbacks, i.e., index server would become a bottleneck when the server usage is high. Some other disadvantages of this architecture are the vulnerabilities on censorship, malicious attack and technical failure.

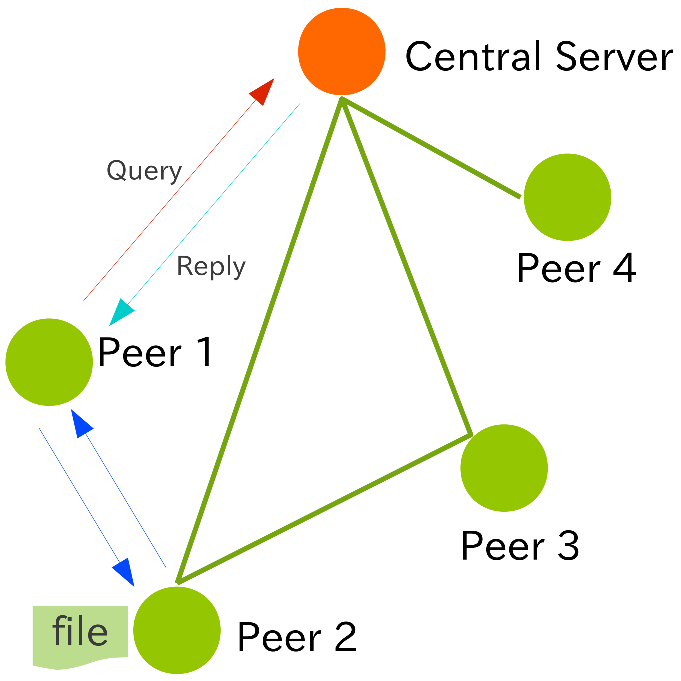


Figure 1.2: Architecture of Napster

- **Pure type**

  Each peer maintains an index of its local files.

  Gnutella [2] is a fully decentralized P2P system, in which each peer can identify the location of shared files by flooding queries to the other peers. Figure 1.3 illustrates the architecture of Gnutella and its search mechanism. There is no central server in the network and user peers connect to each other directly.

Although such a flooding-based approach removes the bottleneck in the centralized approach, it causes several critical issues in realizing efficient and effective file search, such as congestion on low bandwidth peers, high amount of query traffic on the network, and an inaccuracy of the search result.

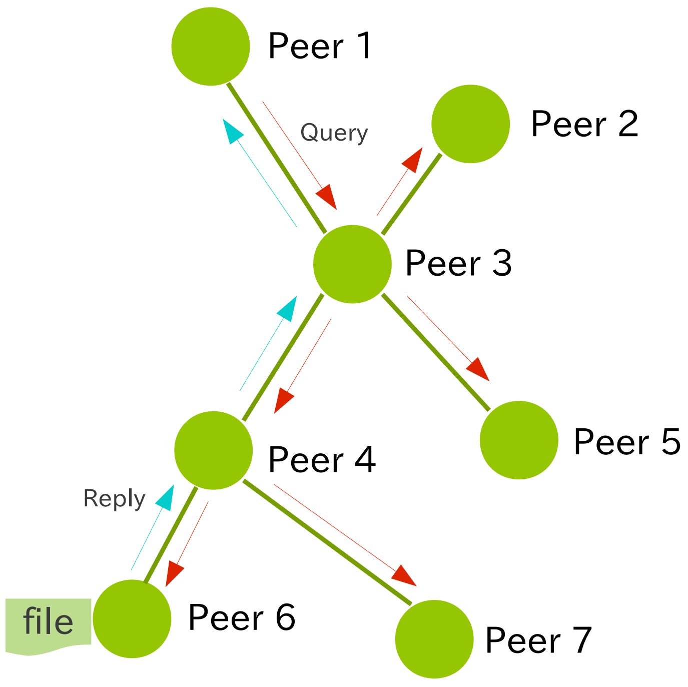


Figure 1.3: Architecture of Gnutella

By control policy of indexing, pure P2P can be again classified into **unstructured P2Ps** and **structured P2Ps**.

– **Unstructured P2Ps**

  In unstructured P2P networks such as Gnutella and FreeNet, the location of files is completely unrelated to the overlay topology. Indices will be retrieved by TTL (time-to-live) based controlled flooding over peers in the systems, i.e., there is no tight control on index of the relevant files. Therefore, query forwarding is totally based on randomization or probabilistic, a large number of traffic will be generated for locating files. Usually, this kind of systems can easily accommodate a highly transient peer population, while the overhead of searching is huge.

  - **Structured P2Ps**

    Alternatively, most of structured P2Ps adopt Distributed Hash Table (DHT), to overcome such a low scalability of flooding-based indexing schemes. Example of DHT systems are Chord [82], Pastry [73], and CAN [69]. In such networks, files are placed at precisely specified locations since it provides a mapping between the file identifier and location, in the form of DHT, so that queries can be efficiently routed to the peer with the desired file.

    However, extra cost for maintaining the structure overlay required for routing is needed, and such maintenance cost of overlay is high when the peers are joining and leaving the network at a high rate. For example, BitComet [10] uses a structured network for storing peer contact information for "trackerless" torrents. Such structured network is based on Kademila [49] and is implemented over UDP. It is reported that in the first 30 minutes after enabling the DHT network, there is a high traffic volume incurred by overlay reconstruction. Therefore, in this sense, the adoption of structured P2Ps are more suitable for a network with stable peers such as super-peer layer in the hierarchical P2P systems (See **Hierarchical type** for the details).

- **Hierarchical type**

  Index of files in a cluster are maintained by its super-peer, and the whole index of files are collaboratively managed by all super-peers.

  The basis is the same as with pure type. However, in order to handle heterogeneity of peers, some of peers are selected to be index servers of files shared by ordinary user peers. These peers are called "super-peers" [26, 25, 50, 53, 61, 95, 98, 103]. Hierarchical P2Ps such as Kazaa [45], which are based on the notion of super-peers, have attracted considerable attentions in recent years. Figure 1.4 illustrates the architecture of Kazaa. Kazaa consists of two layers; i.e., the upper layer consisting of super-peers and the lower layer consisting of ordinary user peers (See Figure 1.4). The bottleneck in the centralized scheme could be certainly overcome by replacing the (single) index server by a collection of super-peers, and the low efficiency of the fully decentralized scheme could be overcome by introducing an upper layer of super-peers. In Kazaa, flooding protocol is used for forwarding queries in super-peer layer. Many other search protocols, such as Chord [82] and Pastry [73] can be applied to super-peer networks.

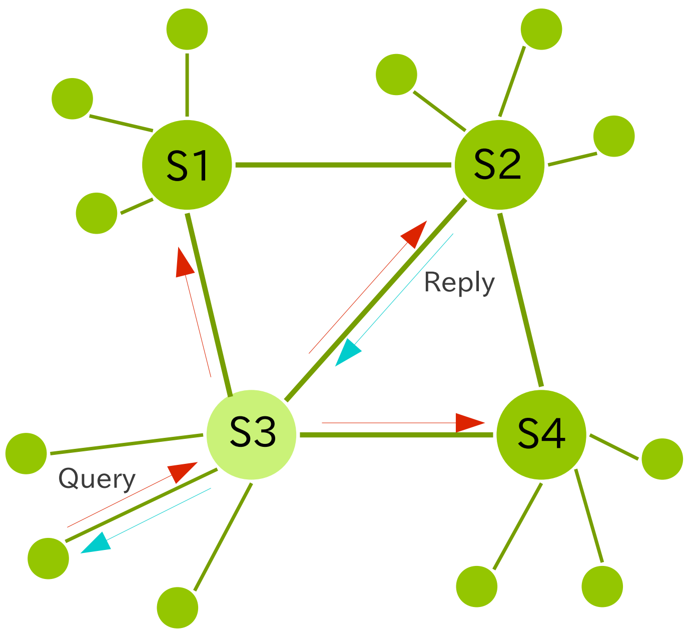


Figure 1.4: Architecture of Kazaa

1.2.2 Search Algorithms

The objective of search algorithms is to successfully locate objects while incurring low overhead and delay. Here, it is worth to noting that inappropriate search algorithms would cause load imbalance of peers. For example, a peer always chooses the same neighbor peer to relay incoming queries, and consequently the load of such neighbor peer would become bigger than other neighbor peers of the peer.

Object Search in Unstructured P2Ps

Search schemes for unstructured P2Ps can be classified into two categories, i.e., blind search and informed search [43, 84, 81]. In a blind search, peers do not keep information about object location. Blind search schemes often employ flooding techniques to relay queries in the network [2, 9, 29, 30, 38, 37]. In an informed search, peers gather some metadata to assist the search operation [86, 96, 97]. The metadata could be the statistics information related to past queries or location information of objects.

Breadth First Search (BFS) is a basic object search scheme adopted in the original Gnutella [2], in which the originator of a search process floods a query message to all peers within a predetermined TTL hops, and it collects reply messages from those peers indicating whether or not a requested file is held by them. BFS generates enormous amount of overhead by contacting multiple peers, low bandwidth peers may simply become a bottleneck of query processing. Random Breadth First Search (RBFS) [38] tries to reduce the overhead of BFS by restricting the number of receivers of a query to a predetermined fraction of the number of neighbors. The portion of neighbors that are chosen is a system parameter of RBFS. In Normalized Flooding [30], a peer of small degree forwards a query to all his neighbors, while a peer of large degree forwards a query to a small subset of its neighbors chosen uniformly at random. Let σ be a threshold of minimum degree in the network. If a peer has more than σ neighbors, then it sends the message only to σ peers that are selected uniformly at random. With normalization, it is easier to control how many peers will be reached by the flooding, thus, searching can be conducted at a finer granularity than BFS.

Random Walk [9, 29, 37] is to forward a query along a randomly constructed path. The basic idea of a random walk algorithm is that when a peer issues or receives a query, each randomly selects a neighbor to send or forward the query. The query message in this process is called a "walker". There are variants of random walk methods. One form involves the use of k-walkers [9, 29]. In k-walker random walk, an originator sends out k query messages to randomly selected k neighbors, and during a search process, each message follows its own random search path. Such search process terminates with a success or a failure; i.e., after finding a target file or after exhausting the TTL. Another variant of random walk is a two-level random walk [37]. An originator selects $k1$-walker random walk with TTL1=$l1$. When the TTL1 timer expires at a particular peer, each walker will then generate $k2$ walkers which will perform $k2$-walker random walk from that peer with TTL2=$l2$. Given the equal number of walkers, this scheme generates less duplicate messages, thereby reducing the system load. On the other hand, it has an increase in the average length of the discovered path.

Directed Breadth First Search (Directed BFS) [96] aims to improve the quality of search results and reduce search overhead. In Directed BFS, each peer maintains information about its neighbors, including the number of results that were received through the neighbor for past queries or the latency of the connection with that neighbor. Since queries are transmitted through a small subset of neighbors, the number of query messages are significantly reduced. In the meantime, an originator is able to choose "good neighbors" to send the query, the quality of query results can be improved.

Adaptive Probabilistic Search (APS) [86] is an improvement of the k-random walk. In this scheme, it utilizes feedback from previous searches to probabilistically guide future ones. Each peer maintains a table for the forwarding probability to each neighbor for each object. The value of each entry in the table mirrors the relative probability of this peer's neighbor to be selected as the next hop in a future incoming query for a specific object. APS employs k-walker random walker to search for the required object and each intermediate peer forwards the query to one of its neighbors with a probability given by its table index. If a walker succeeds, the relative probabilities of the peers on the walker's path are increased. Each peer updates its local table according to the search results, so as to reflect an actual distribution of objects in the network to the collection of local tables. APS algorithm shows improved performance over the random walker model.

In Local Indices [97] search method, each peer maintains indices of all peers located within a predefined hop distance. Therefore, each peer can directly answer queries for any objects in its local indices without resorting to other peers. The purpose of this method is to reduce the number peers processing a query. Although local indices approach reduces the processing cost by limiting processing to fewer peers, it incurs a higher update cost for these indices. Due to the dynamics of the network, the indices may be obsolete or inconsistent.

Object Search in Structured P2Ps

Object search in structured P2Ps is conducted in a more systematic manner compared with unstructured P2Ps. Most of structured P2Ps adopt DHTs. DHTs are often referenced by their geometry of hash space, such as ring [82], torus [69], tree [59] and some hybrid [73][101].

In Chord [82], peers and keys are assigned an m-bit identifier using consistent hashing such as SHA-1 algorithm. Using Chord protocol, peers are arranged in a ring that has at most 2^m peers. The ring can have keys ranging from 0 to $2^m - 1$. Each peer has a successor and a predecessor. The successor to a peer (or key) is the next peer (key) in the identifier circle in a clockwise direction. The predecessor is in a counter-clockwise direction. If each peer knows only the location of its successor, a linear search over the network could locate a particular key. This is a naive method for searching the network, since any given query message could potentially have to be relayed through most of the network. To accelerate lookup, Chord implements a faster search method by maintaining additional routing information. Chord requires each peer to keep a "finger table" containing up to m entries. The i^{th} entry of a peer n contains the address of successor $((n + 2^{i-1}) \mod 2^m)$. With such a finger table, the number of peers that must be contacted to find a successor in an N-peer network is $O(\log N)$. Figure 1.5 illustrates routing algorithm in Chord. In Figure 1.5, the ring can have keys ranging from 0 to 7. The overlay consists of 4 peers, each of which maintains a finger table consisting of 3 entries. Peer 1 issues a query for searching key 7. By referring its finger table, the query will forwarded to Peer 6, and then forwarded to the Peer 0.

Pastry [73] realizes a message routing and an object allocation in potentially very large overlay networks [73]. The special design of Pastry compared to other P2P systems is that Pastry considers the network locality in message routing. The purpose of this design is to minimize the network distance of messages in traveling. Pastry employs a prefix-based routing scheme where each peer has a 128-bit identifier chosen by MD5, and maintains a routing table consisting of a leaf set, a neighborhood set, and a routing table. The leaf set consists of the $L/2$ closest peers in each of directions. The neighborhood set contains M closest peers in terms of the proximity metric. The neighborhood set is not normally used in routing messages, while it is useful in maintaining locality properties in the routing table. The routing table is organized into $\log_{2^b} N$ rows with $2^b - 1$ entries each, where b is the parameter of splitting a

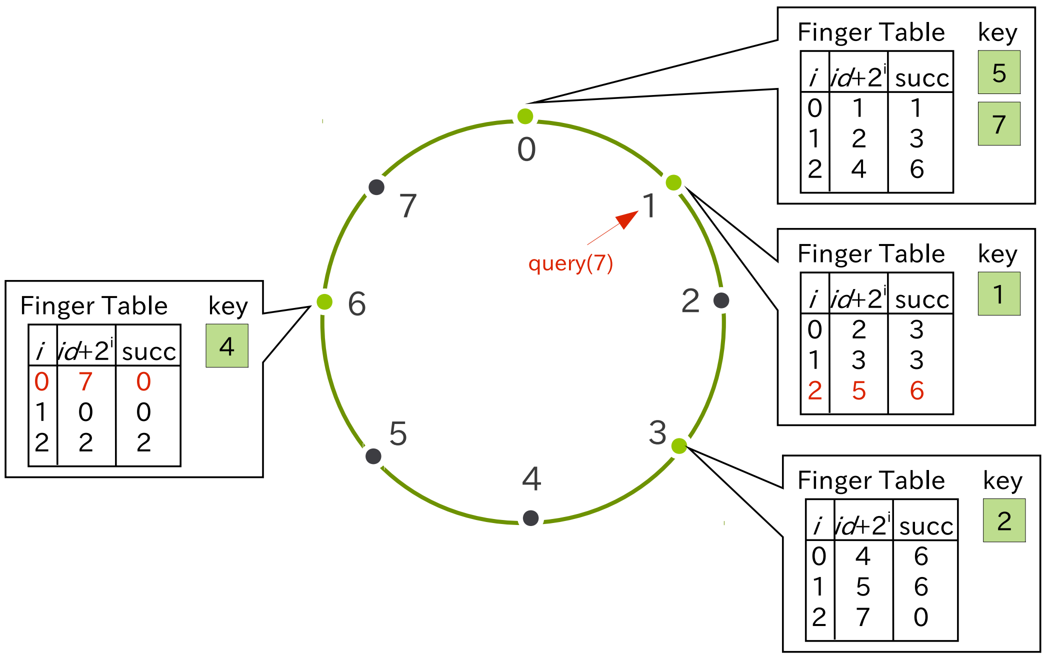


Figure 1.5: Routing of Chord

key into digits. The entries at row i point to peers that share the first i digits of the prefix with the key. Given a query, the originator first checks if the key is within the leaf set. If so, it forwards the query to the closest peer in the leaf set. Otherwise, Pastry forwards the query to a peer with one more matching digit in the common prefix. In the rare case, when Pastry cannot find a peer that matches the first two criteria, it forwards the query to any peer that is closer to the key than the current peer's identifier. Figure 1.6 illustrates routing algorithm in Pastry. As shown in figure 1.6, each peer has a 4-bits identifier. when a key “d3b8” are given to the network, it efficiently route the message to the destination peer based on prefix routing, i.e., “$53a1 \rightarrow d152 \rightarrow d3a0 \rightarrow d3b1$”.

CAN [69] centers around a virtual d-dimensional Cartesian coordinate space on a d-torus. Consider an object held by a peer in the system. To realize an efficient management of the object, CAN stores a key-value pair $(K1, V1)$ to the virtual co-

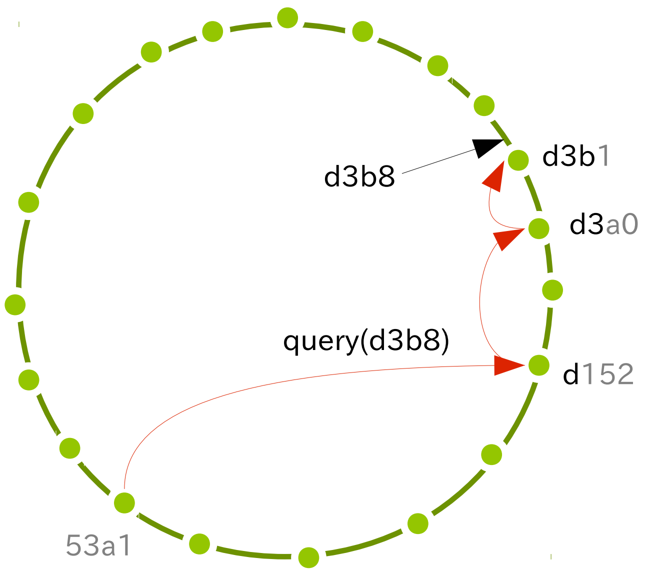


Figure 1.6: Routing of Pastry

ordinate space, where $K1$ corresponds to the name of the object and $V1$ represents the name of the peer that holds the object. CAN first maps key $K1$ onto a point in the coordinate space in a deterministic manner by using a uniform hash function. It then stores the corresponding pair at the peer that owns the zone within which the point lies. The retrieval of a stored pair could be done in a similar manner; that is, to retrieve an object corresponding to key $K1$, any peer can apply the same deterministic hash function to map the key onto the target point and then retrieve the corresponding value from the point. Figure 1.7 illustrates routing algorithm in a 2-dimensional coordinate space of CAN.

Each of the above search schemes for structured P2Ps tightly controls object allocation and the topology of the overlay network in order to realize certain kind of search ordering to facilitate the search of requested files. However, although it would certainly improve the efficiency of the file search process, such a tight control leads to a high overhead in the data allocation and the topology maintenance. The search

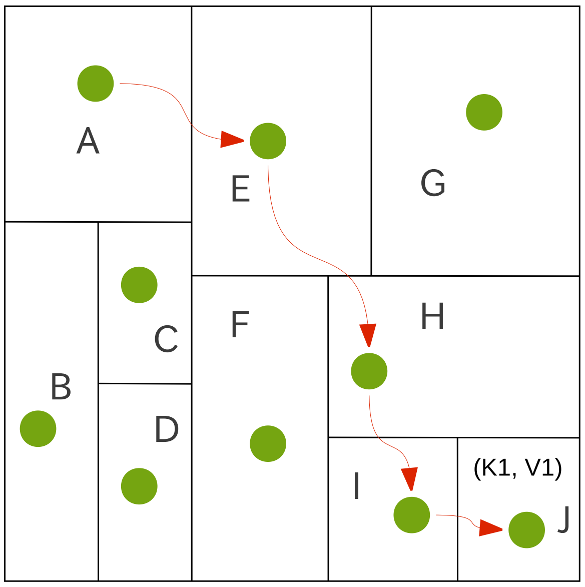


Figure 1.7: Routing of CAN

performance of structured P2Ps will be degraded when routing information becomes out of date due to dynamicity of peers.

1.3 Research Aim and Approach

From a technological point of view, there is nothing a P2P system can offer, which cannot be implemented with a centralized system [78]. Especially in nowadays, when users are used to nearly instant reply backed up by a data center or a Cloud, P2P systems cannot impress users with response time of queries. Nevertheless, Web systems such as Google [33] and Yahoo [93] are developing more distributed architectures and algorithms to manage with the scalability issue [5], there is still a lot of opportunities left for P2P systems. For example, Faroo [23] and YaCy [92] are two popular distributed information retrieval systems (IR systems) based on P2P model.

In this thesis, the reason we focused on P2P systems is that it has some important properties compared to Web systems; i.e., 1) Pull-based (crawler-based) index collection mechanism in Web systems causes an inevitable delay in reflecting the change of files to the collected indices. While P2P systems would naturally use push mechanism: peers decide themselves which files they want to make searchable. 2) Search technology is not transparent and comprehensible in closed Web systems. While in P2P systems, peers can determine how to index files. For example, a peer can associate tags to a file for describing the classification of the file or the personal interest of the peer.

In contrast to Web systems, P2P systems impose many serious restrictions on information retrieval such as [78]:

- **Decentralization.** No centralized units are available.
- **Heterogeneity.** Bandwidth, connectivity and capacity of peers may highly varied.
- **Dynamicity.** Churn rate is substantially higher than the failure rate in Web systems.
- **Limited connectivity.** Stability of overlay connections is often worse than connections in server cluster networks.

To overcome such a difficulty, the use of *super-peers* is currently being the most promising approach in optimizing allocation of system load to peers, i.e., allocate more system load to high capacity and stable super-peers by assigning task of index maintenance and retrieval to them. In this thesis, we focused on two kind of super-peer based *hierarchical architectures* of P2P systems, which are distinguished by the organization of super-peers [25, 61]. In each of them, we discussed system load allocation, and proposed novel load balancing algorithms for alleviating load imbalance of super-peers, aiming to decrease average and variation of query response time during index retrieval process.

More concretely, in this thesis, our contribution to *load management* solutions for hierarchical P2P file search are the following:

- Qin *et al.* proposed a super-peer based three-tier hierarchical architecture for real-time file search [61]. Figure 1.8 illustrates the architecture of Qin's system. In this architecture, indices of files held by the user peers in the bottom layer are stored at the super-peers in the middle layer, and the correlation of those two bottom layers is controlled by the central server(s) in the top layer using the notion of tags. In Qin's system, a heavily loaded super-peer can move excessive load to a lightly loaded super-peer by using the notion of task migration. However, such a task migration approach is not sufficient to balance the load of super-peers if the size of tasks is highly imbalanced. To overcome such an issue, in this thesis, we propose two load-balancing schemes for this architecture, aiming to ensure an even load distribution over the super-peers. The first scheme controls the load of each task in order to decrease the total cost of task migration. The second scheme directly balances the load over tasks by reordering the priority of tags used in the query forwarding step. The effectiveness of the proposed schemes are evaluated by simulation. The result of simulations indicates that all the schemes can work in coordinate, in alleviating the bottleneck situation of super-peers.

- In a two-layer DHT-based super-peer architecture, Distributed Hash Table (DHT) is adopted into super-peer layer to collaboratively manage indices of files held by the user peers [25]. Figure 1.9 illustrates the architecture of DHT-based super-peer system. In this architecture, the skewness of user's preference regarding keywords contained in multi-keyword query causes query load

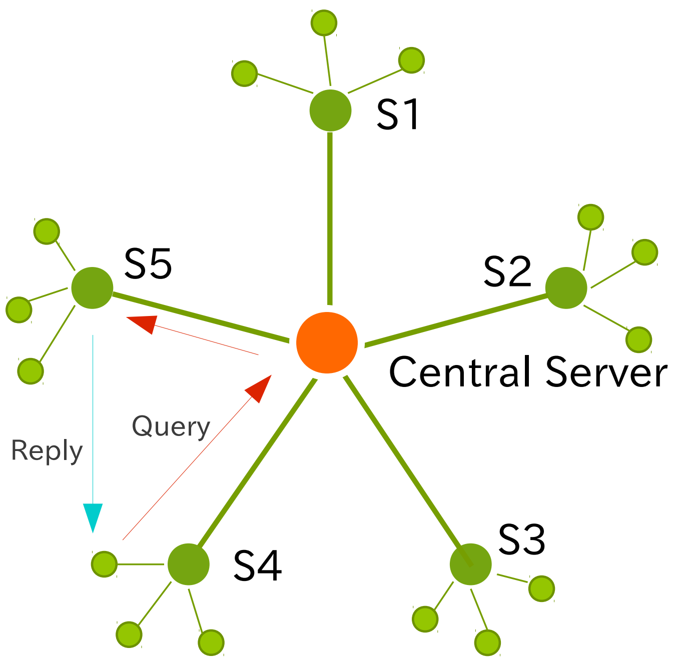


Figure 1.8: Architecture of Qin's hierarchical P2P system

imbalance of super-peers that combines both routing and response load. This imbalance means long file retrieval latency that negatively influences the overall system performance. Although index replication has a great potential for alleviating this problem, existing schemes did not explicitly address it or incurred high cost. To overcome such an issue, in this thesis, we propose an integrated solution that consists of three replication schemes to alleviate query load imbalance while minimizing the cost. The first scheme is an active index replication in order to decrease routing load in the super-peer layer, and distribute response load of an index among super-peers that stored the replica. The second scheme is a proactive pointer replication that places location information of an index, for reducing maintenance cost between the index and its replicas. The third scheme is a passive index replication that guarantees the maximum query load of super-peers. The result of simulations indicates that the proposed schemes can help alleviating the query load imbalance of super-peers. Moreover, by comparison it was found that our schemes are more cost-effective on placing replicas than other approaches.

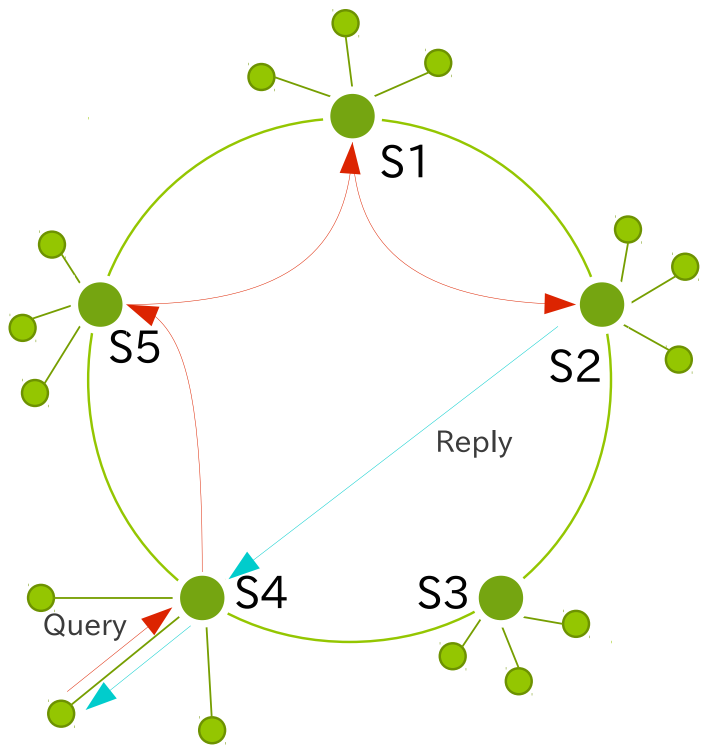


Figure 1.9: Architecture of DHT-based super-peer system

1.4 Thesis Organization

Chapter 1 provides a broad overview of this thesis. Firstly, we introduced load balancing problem in parallel and distributed computing, Web server systems and P2P systems. Load balancing is one of fundamental technologies in improving performance of these systems. Secondly, current P2P architectures and search algorithms are examined with a focus on load management in the systems. We clarified that load management of P2P systems is a wide issue, and it has to be solved from both architectures and algorithms. Thirdly, we pointed out that the main work of the thesis is to study the load management for two kinds of hierarchical P2P architectures, i.e., Qin's hierarchical P2P architecture and DHT-based super-peer architecture. The scope and significance of the thesis are discussed.

In Chapter 2, we present the state of the art of load balancing in P2P systems. We classified existing solutions into two types, i.e., task migration based approach and task replication based approach.

In Chapter 3, Qin's hierarchical P2P architecture is described in detail. We propose efficient task migration schemes for load balancing of super-peers.

In Chapter 4, DHT-based super-peer architecture is described in detail. We propose cost-efficient task replication schemes for load balancing of super-peers, as well as routing load reduction of super-peers.

In Chapter 5, we summary and give an outlook on future work of this thesis.

Thus, the contributions of this thesis are:

i) Analyzed load problem and designed load metric in dedicated hierarchical P2P systems;

ii) Proposed efficient and effective solutions to overcome load problems in these systems.

Chapter 2

A Survery on Load Balancing in P2P Systems

2.1 Introduction

In general, load balancing is a key technology for avoiding a peer failing in performing its tasks in distributed systems. Load balancing in P2P systems has been researched for more than ten years since its appearance. Load balancing can be achieved by shedding load from heavily loaded peers (i.e., source peer) to lightly loaded peers (i.e., destination peer) via task migration or task replication. Unlike replication, migration implies that once task has been moved to a destination peer, it will be deleted at the source peer.

Apart from heterogeneity of peers, there are many causes of load imbalance of peers in P2P systems. Serbu [74] pointed out that load problem in DHTs may occur in four resources, i.e., overlay namespace, requests, routing and underlying topology, and such imbalance may occur in all resources. In this chapter, we give a brief survey on load balancing schemes for P2P systems. In the survey, we focus on load problem related to the P2P system itself, i.e., we do not consider the underlying network

Table 2.1: Causes of load imbalance

Resource	Description	Notion of load
Identifier space	uneven identifier intervals assigned to peers	storage/response load
Indegree size	uneven in-links associated to peers	query load combining of response load and routing load
Response	skewed queries towards to popular keys	response load
Routing	biased utilization of links for routing query	routing load

management. Based on Serbu's work [74], we summarize the main causes of load problem in P2P systems in table 2.1, details of each cause is listed as follows.

- **Identifier space** In DHTs, all peers and objects share an identifier space. An unequal identifier interval assignment would likely to generate non-uniform number of keys assigned to peers. Typically, a key means a query or an index entry of an object.

- **Indegree size** In unstructured P2P, indegree size of peers usually follows power-law phenomenon, i.e., a few peers has bigger indegree sizes than other peers. The high indegree peers have would likely to forward and receive more queries than other peers.

- **Response** Popularity of keywords contained in queries often follow Zipf's law, i.e., some keywords (or keys) are popular, being requested much more often than others. Thus, the peers that are responsible for popular keys have to respond more queries than other peers.

- **Routing** When a large number of queries are passing through a peer, the peer and its corresponding links are likely to become overloaded. This situation may be incurred by inappropriate routing algorithms of P2P systems.

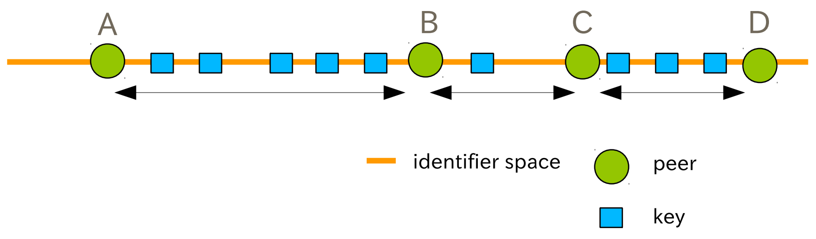


Figure 2.1: Load imbalance occurred in identifier space

Figure 2.1 illustrates load imbalance occurred in identifier space in DHTs. In figure 2.1, the identifier space is represented as a line. We assume that each peer is responsible for the keys in its left interval. As shown in figure 2.1, B is responsible for the largest identifier interval, and consequently it is associated with more keys than C and D. C and D have equal identifier interval, while D is associated with more keys than C. It explains that load imbalance may still occur when the assignment of keys to peers is non-uniform, even if identifier interval of peers are balanced.

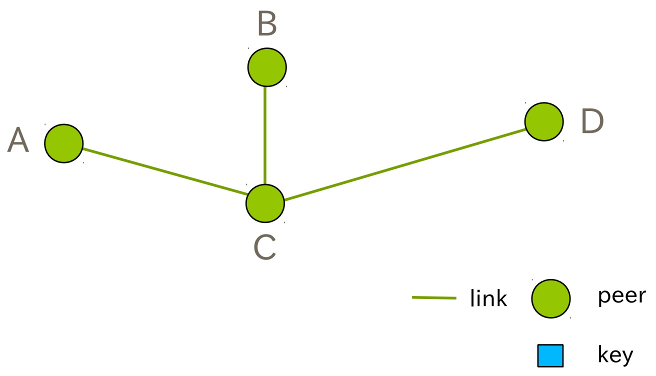


Figure 2.2: Load imbalance occurred in indegree size

Figure 2.2 illustrates load imbalance occurred in indegree size in P2Ps. In this figure, C has larger indegree size than other peers. In power-law networks, the high degree peers tend to experience more traffic load than low degree peers [1]. In light of this principle, an elastic routing table (ERT) mechanism is also presented in DHTs [76]. Unlike routing tables with a fixed number of outlinks in current DHTs, each ERT has a different number of inlinks/outlinks, and the indegree/outdegree of each peer can be adjusted dynamically according to its experienced traffic so as to direct traffic flow to lightly loaded peers.

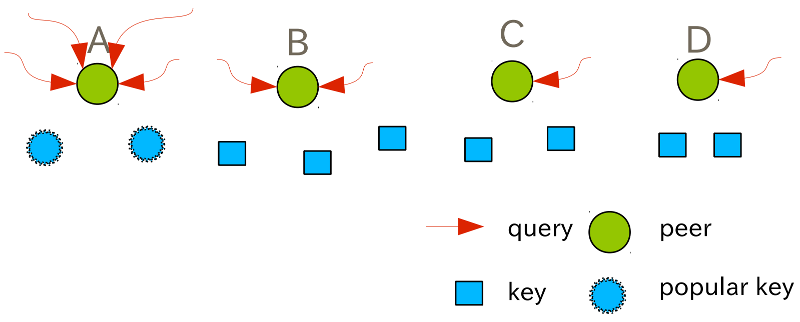


Figure 2.3: Load imbalance occurred in response

Figure 2.3 illustrates load imbalance occurred in response in P2Ps. The response load of a peer is related to not only the number of keys (or objects) stored at it, but also the popularity of the keys (or objects). In figure 2.3, A, C and D are all responsible for two keys (or objects). But, A owns two popular keys (or objects). Thus, A will receive more queries than C and D. On the other hand, B owns three keys (or objects), thus B may potentially receive more queries than C and D.

Figure 2.4 illustrates load imbalance occurred in routing in P2Ps. In figure 2.4, most of queries between A and D, prefers to pass by B rather than C. This may be occurred by two causes: 1) routing algorithm such as A usually selects B to forward queries. 2) overlay structure such as B is in the intersection of multiple routing paths.
]

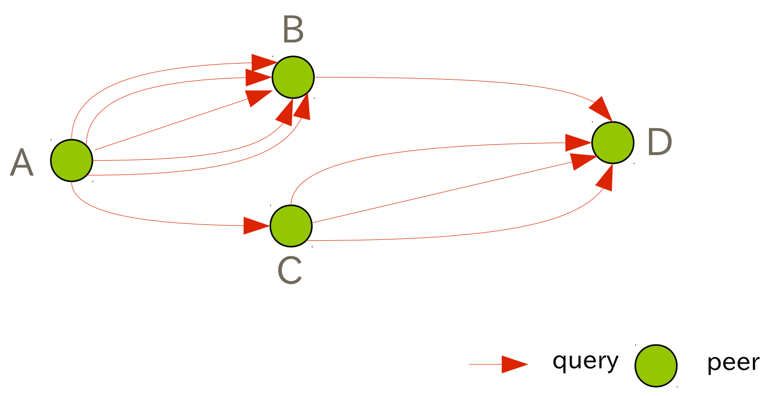


Figure 2.4: Load imbalance occurred in routing

Measurement of the "value" of load on a peer in P2P systems is not an easy task. The load on a peer may vary greatly over time due to dynamism in the following forms [31]: (1) continuous insertion and deletion of objects, (2) skewed object arrival patterns, and (3) continuous arrival and departure of peers. Moreover, the query pattern and request rate of user peers may change over time [75]. In query type of keyword search, the popularity of keyword may follow uniform distribution where each keyword has the same possibility attached into a query, Zipf's distribution where some keywords are much more popular than others, or some kind of distribution generated by real users. Thus, load balancing solutions have to be adaptive in P2P systems.

2.2 Task Migration based Approaches

we overview task migration based load balancing for each type of P2P architectures; i.e., structured type and unstructured type.

2.2.1 Structured P2Ps

Most of structured P2Ps adopt Distributed Hash Table (DHT) as an overlay network, to realize an efficient data management and information retrieval. In such P2Ps, which are referred to as P2P DHTs in the literature, each object typically attached several keys, and each key is mapped to a unique point in a hash space. The hash space is partitioned among peers in the P2P, and each peer is responsible for storing all indices of objects whose keys are mapped to a portion of the hash space associated to the peer. Each peer can efficiently locate a peer responsible for a given key, by identifying a point corresponding to the key in the hash space and by routing a query message towards the identified point. P2P DHTs generally assume that all peers are homogeneous, objects are of the same size, and IDs given to all objects are uniformly distributed over the ID space (i.e., key space) of the DHT. Examples of P2P DHT include Chord [82], CAN [69], Tapestry [101], and Pastry [73].

At an early research stage, load balancing in DHTs aimed to balance storage load of peers, i.e., the number of objects per peer. CFS [20] introduces the notion of *virtual servers* which play a similar role to the tasks in conventional load balancing schemes. More concretely, each peer is assigned several virtual servers each of which is responsible for a portion of the ID space, and a load distribution is realized by migrating virtual servers among peers. CFS improves load balancing of peers by taking into account the peer heterogeneity, i.e., it determines the number of virtual servers associated to a peer in such a way that it is proportional to the peer capacity. CFS also introduces a mechanism to decrease the number of overloaded peers by allowing each overloaded peer to “drop” some of its virtual servers. Rao *et al.* [67] propose three schemes in which virtual servers are migrated from a set of heavy peers to a set of light peers. As such, the notion of virtual servers makes it easy to realize an effective load balancing in P2P DHT. However, it has a serious drawback such that it should use a number of *directories* to store and update the load information of the peers in the system, which consumes a large amount of resources in P2P systems. In order to reduce the high maintenance cost of multiple virtual servers, Godfrey and Stoica [31] propose a simple DHT protocol called Y_0. In Y_0, a peer clusters a number of virtual servers in a random fraction of ID space, instead of selecting virtual servers with random IDs. Therefore, the peer could share a single set of overlay links among a cluster of virtual servers and create one routing table for the whole cluster. Further, by modifying successor list, figure table and routing algorithm, Y_0 achieves good load balancing with a sufficient low overhead.

Another part of load balancing schemes is realized by controlling the location of objects or peers. Rieche *et al.* [70] proposed load balancing schemes based on moving objects between peers. In the proposed schemes, the ID space is divided into intervals. Peers within an interval are collaborative for all of the objects stored in the interval. If the storage load in an interval exceeds a threshold value, the following operations are conducted; i.e., 1) the interval with excessive load is split into two intervals if the interval contains more than $2f$ peers, 2) it move peers from other intervals to the overloaded interval to reach $2f$ peers and split it as previous case, 3) if there are no more than f peers within the interval, interval borders between the overloaded interval and its neighbor intervals can be shifted to balance the load. Karger and Ruhl [39] proposed a mechanism that allows lightly loaded peers to relocate to the ranges of the ID space associated with many objects, in order to share the load of the peers responsible for such ranges of DHT space. More concretely, a lightly loaded peer requests the load of a randomly chosen remote peer and makes a load comparison between itself and the remote peer. If the load of the remote peer is higher, the peer will relocate to share the range of ID space of the remote peer. However, it is not evaluated under adversarial environment.

As an alternative approach, Byers *et al.* [12] proposed a simple but efficient load balancing mechanism , i.e., power of two choices. When inserting an object into the system, the object are generated into multiple hash values using multiple hash functions, each of which corresponds to a candidate peer of receiving the object. After retrieving the load of the corresponding peer candidates, the object will be stored at the peer with the lightest load. Conversely, when searching an object, the requester should issues multiple queries, for each one using a different hash function. The power of two choices state a quite surprising result, i.e., it was found an exponential improvement in reduction of the maximum load of peers by using two hash values, and further only a constant improvement when using more than two hash values.

Ganesan *et al.* [24] propose a method based on the skip graph [4] which is a distributed data structure to store an ordered set. It keeps a vector representing the load of peers in the form of a skip graph, and updates the vector upon detecting the change of the load. The main drawback of this scheme is that it incurs a substantial cost for maintaining complete information about the load at every peer in the system. Jagadish *et al.* [36] propose a scheme based on the notion of heap, which maintains a set of light peers using heap (i.e., a balanced binary tree of peers) so that each peer can acquire a list of light peers by simply referring to the heap. At last, Vu *et al.* [88] propose a general framework HiGLOB based on structured P2Ps and demonstrate the effectiveness by instantiating it over three existing structured P2P systems: Chord [82], Skip Graph [4] and BATON [36]. HiGLOB employs histograms to keep the load distribution of the system. Each peer constructs a load histogram by partitioning the network into non-overlapping groups. If it detects load imbalance, it performs load exchange similar to object balancing algorithm [39].

At a later research stage, some of researchers focused on balancing routing load dealing with skewed popularity of objects. Bianchi *et al.* [8] proposed an adaptive load balancing mechanism in Pastry; i.e., replacing peers with high traffic rate to peers with low traffic rate in the routing tables if a peer becomes overloaded. Shen and Xu [76] proposed an load balancing scheme based on elastic routing tables. By resizing out-degree (i.e., size of routing table) of peers, thereby consequently controlling the in-degree assignment of peers, it can balance routing load of peers.

2.2.2 Unstructured P2Ps

In unstructured P2Ps such as Gnutella, random peer selection is commonly used to realize a load migration. Chawathe *et al.* [16] propose a scheme based on the notion of random walk. It tries to increase the probability of visiting a light peer by organizing the overlay in such a way that peers with a high capability have a large number of adjacent peers, while it would need a global view on the distribution of peer capabilities. Bharambe *et al.* [7] propose a protocol Mercury using histogram to maintain the load distribution in the system. In Mercury, a peer maintains its histogram by periodically sampling nearby peers to get local load and exchanging local load information with far away peers based on random walk algorithm.

Garbacki *et al.* [26] propose a load balancing scheme for hierarchical P2Ps, based on a list of super-peers (SPs) called super-peer cache. In this scheme, an overloaded SP reduces the number of requests received from ordinary peers by decreasing its priority in the super-peer cache, where each ordinary peer tries to connect to SPs in the order given in the super-peer cache. Although it could autonomously reduce the load of overloaded SPs, it has a serious drawback such that selfish SPs would significantly degrade the performance of the overall system, since it lacks a mechanism of determining the relative load of SPs. In hybrid P2Ps such as Napster and eDonkey, an aggregation of global load information could be realized by using an index server as an information aggregator. However, such a central server easily becomes a bottleneck as increasing the number of participating peers, if we try to collect "all" of the latest information to the aggregator.

2.3 Task Replication based Approaches

Replication is widely used to improve object availability[66, 57, 46] and search efficiency [65][68][75]. In a system, by storing a data at more than one peer, even if a peer fails, the system can operate using replicated data, thus increasing availability and fault tolerance of the system. At the same time, as the data is stored at multiple peers, a request can find the data close to the peer where the request originated, thus increasing the performance of the system. However, the benefits of replication come with overheads of creating, maintaining and updating the replicas. If the data is read-only, replication can greatly improve the performance. But, if the data needs to be updated, the overhead of maintaining consistency among multiple replicas would become a critical issue, as will be discussed in the Section 4 of the thesis. In P2P systems, replication also implicitly improves the load balancing of peers, since replicas in the intermediate peers can intercept the queries towards home peers of popular objects, the load of home peers is shared by replica peers, and a part of peers' routing load is also decreased.

By "candidate peer" of replicas, replication schemes can be categorized into three types; i.e., client-side replication, server-side replication and path replication. By given an object, client-side replicates the object close to requester; server-side replicates the object close to object owner and path replicates on the peers in the query path from a requester to the object owner.

By "data" of replication, replication can be classified into file replication, index replication and pointer replication. File replication replicates the file to other peers. The management cost of file replicas is not trivial if the size of files are large. Index replication replicates index of files to other peers. Index contains meta information of files, and can be used for responding queries. The management cost of index replicas is much smaller than object replication, but the maintenance cost of index replicas may be high if we want to guarantee the consistency of search results. Pointer replication replicates a pair of a given key and the responsible peer that has the keyword. Pointer would be an effective approach to avoiding the consistency issue of index replicas.

When designing a replication strategy, the following important aspects should be taken into consideration [57]:

1. What to replicate (replica selection problem)?
2. Where to place the replica (replica placement problem)?
3. When and how to update the replica (update control problem)?

Keeping these questions in mind, we overview task replication based load balancing for each type of P2P architectures; i.e., structured type and unstructured type.

2.3.1 Structured P2Ps

In structured P2P, a key-value pair is replicated to several peers other than responsible peer. By this replication, if later coming queries reaches these replica peers, the requested value can be returned by these replica peers.

Bianchi *et al.* [8] consider the load problem by lookup traffic. They presented an analysis of structured P2P systems taking into consideration Zipf-like requests distribution, i.e., few highly popular objects being requested most of the times. In their study, they consider the load as the number of received and forwarded requests per unit of time. In the simulation, they confirmed the load imbalance problem, and use server-side replication in attempt to reduce the request load of peers.

Li *et al.* [44] presented two replication schemes to balance routing load and query load. The first scheme is to replica popular keys from heavily loaded peers to lightly loaded peers via broadcasting protocol. Replicating popular keys relieves the query answering load of the peers responsible for the keys. However, a peer may be still be overloaded caused by the traffic of forwarding incoming routing queries. The second scheme is to share routing load by copying routing table to a lightly loaded peer. Besides load balancing, it can also improve the routing resiliency.

Ghodsi *et al.* [28] presented symmetric replication which only needs $O(1)$ message for every join and leave operation to maintain any replication degree. The main idea behind symmetric replication is that each identifier in the system should be associated with f other identifiers. If identifier i is associated with identifier r, then any item with identifier i should be stored at the peers responsible for identifier i, and r. Similarly, any item with identifier r should also be stored at the peers responsible for the identifiers i, and r.

Ramasubramanian and Sirer [65] proposed an approximate replication solution for Pastry. Replicas of a given object is placed on a set of peers who has the same digit-based prefix of peer ID. By estimating the popularity of each objects, different ranges of ID space for each objects is determined. The solution is dedicated to Domain Name System (DNS), where each object is a static URL. The popularity of URLs is generally relatively stable, while popularity of objects in file sharing systems is time-varying.

Rao *et al.* [68] proposed an optimal replication algorithm (referred to as PCache in this thesis) for minimizing average search hops. The algorithm is based on the observation that peers with smaller overlay distance to object owner have higher visit frequency, thus the replicas are suggested to place consecutively at predecessor peers. Each replica peer will later respond to lookups for the object. However, in fact, their observation might not be true under the most common cases, i.e., some of predecessor peers close to object holder might forward very few queries if the access pattern over requesters is skewed. Hence, the replicas in PCache might not be fully utilized. Rao *et al.* claimed that the number of replicas generated for an object should be proportional to the popularity of the object if the total number of replicas is predetermined, so as to optimize the average search hops. However, the popularity estimation for all objects is a very difficult task in a distributed environment.

In order to improve the hit rate of replicas, Shen [75] proposed an efficient and adaptive decentralized (EAD) replication algorithm. In the algorithm, each peer records and updates query traffic rate of each object, denoted by q_f. By determining a constant value T_q based on the average query rate in the system, if $q_f > T_q$, the peer will initiates an object replication request to the object owner. Such requests will be temporarily saved in a cache of the object owner during a time period. If the object owner goes to overloaded, it will replicate object to the peers in the cache according to q_f in a descending order. EAD can highly improve the utilization rate of replicas. However, in the manner of load shedding, EAD may place replicas close to object owner, thereby EAD is not fully utilized the replica on reducing search hops.

2.3.2 Unstructured P2Ps

Since unstructured P2P systems use flooding or random probing-based methods for file location, the number of replicas directly affects the efficiency of file query. These works study the system performance such as successful queries and bandwidth consumption when the number of replicas of a file is uniform, proportional and square-root proportional to the query rate [48, 18, 83]. In the uniform strategy, replications are uniformly distributed throughout the network. For each data object, approximately the same number of replicas are created. While this controls the overhead of replication, replicas may be found in places where peers do not access the files. In the proportional replication, the number of copies for each object is proportional to its query distribution. The higher the query rate of an object, the higher is the number of copies for that object. On the other hand, with proportional replication, although queries on popular data are processed efficiently, unpopular data search may take a long time, thus degrading the overall system performance. Square-root replication was proposed to provide the optimal search size on successful resource request queries and minimize the overall search traffic.

Another category of replication algorithms is based on selection of destination replica peers. Client-side replication and path replication are two popular methods. In client-side replication, only the sender peer of the query stores a replica of the requested data. In path replication, The requested data is replicated on all the peers along the data transmission path between the peer requesting the data and the peer having the data. These schemes have been employed in many distributed systems because of ease of implementation [17, 72, 48].

Yamamoto *et al.* [94] focused on path replication and developed a way of load balancing on distributed storages in P2P Network. In order to mitigate the load concentration on the high-degree peers over the P2P network without deteriorating the search performance too much, they propose the two methods: Path Random Replication and Path Adaptive Replication. In path random replication, each intermediate peer randomly determines whether or not the replica is created and placed there based on the probability of the pre-determined replication ratio. Path random replication may still cause much load imbalance, because high-degree peers are frequently located in the data transmission path. Therefore, each peer should adaptively determine whether or not to create a replica depending on its resource status. Path adaptive replication further improves the procedure in the decision to make a replica on a peer depends on how much storage is still available on it as well as the predetermined replication ratio.

Kawasaki *et al.* [40] proposed a more effective replica placement method than probabilistic ones. The proposed strategy is cooperated with index caching, as well as LRU-based content replacement. More concretely, a replica will be allocated on the halfway peer between the requester and the holder, as long as the access rate for that content exceeds a threshold value. Each peer has limited capacity of storage, and when its storage is full, a replica is discarded in a LRU (Least Recently Used) manner. The proposed strategy can reduce the excessive traffic and improve search efficiency if the threshold for switching from an index to a replica placement is appropriately determined.

Commercially available super-peer based P2P system such as Kazaa [45] use super-peers in the network to replicate the index of several other peers. It is evident that searching is more effective when a larger index can be probed. However, even though per-probe capacity is increased in these systems, the same resource replication strategy as Gnutella is deployed for the actual replication of resources. Yang *et al.* [95] compare the performance of super peer systems with different replication and server organization schemes. Four different types of architectures are proposed based on server organization, query forwarding and object index replication: Chained Architecture, Full Replication Architecture, Hash Architecture and Unchained Architecture. However, the replication of the actual resources is not addressed.

Rajasekhar *et al.* [64] addresses the problem of load sharing in P2P networks across heterogeneous super-peers, and propose two novel techniques to share the load using replication techniques. In the first technique, called Periodic Push-based Replication (PPR), super-peers periodically send replicas of the most frequently accessed files to remote super-peers. This effectively reduces the hop count to fetch these files. The second technique, called On-Demand Replication (ODR), performs replication based on access frequency. By performing replication on-demand, ODR provides adaptability to changes in access behavior. It is found that a significant improvement in super-peer networks where a reduction in hops result in lower transmission cost, reduced latency and improved QoS.

Rong [71] proposed an adaptive resource replication strategy to facilitate resource replication in a dynamic resource adaptation P2P network. The replication strategy takes an active approach through using resource request rate as the prime metric to trigger the replication process and then adaptively creating resource variations in the network according to the heterogeneities of peers. In addition, the strategy uses peer related information stored on super-peers to determine which peers should be selected to perform adaptive replications and where the resulting replicas should be stored. Simulation results showed that the proposed scheme reduces network delays while increasing resource success rate in comparison to commercial super-peer P2P systems and random replication strategies.

Puttaswamy *et al.* [60] improve the search effectiveness for rare items, and reduce the bandwidth overhead incurred in super-peer based P2P networks. It explores the use of two-hop index replication to significantly improve the effective search space. In the two-hop index replication scheme, each peer sends its index to all of its one-hop neighbors in its routing table. All of the one-hop neighbors, in turn, forward this index to all of their one-hop neighbors except the source peer. This strategy effectively reduces to a two-hop flooding of indices around the peers. Two variants of two hop index replication are used: SR replication and constant replication. In SR replication, each peer performs one-hop replication. Super-peers then replicate the indices of their one hop neighbors to a random subset of their super-peer neighbors. Each super peer's two-hop replica set size is equal to the square root of the number of its super-peer's neighbors. The advantage is reduction in the amount of replication and its cost. In constant replication, each super-peer does two-hop index replication to a constant number of super-peer neighbors. After each peer does one hop index replication, super-peers propagate the index to only a constant number of their super-peers. This reduces indexing load on super-peers.

2.4 Summary and Discussion

In this survey, we have identified the main causes of load imbalance in P2P systems. We have confirmed that load problems could occur in the following resources, i.e., identifier space, indegree size, response and routing. In each resource, the load is represented by a different metric, i.e., storage/response load, query load, response load and routing load. We have then classified the solutions for load problems into two categories, i.e., task migration based approach and task replication based approach. We have presented the most relevant existing solutions in each category.

In a more general sense, there are two types of load balancing solutions: the system-specific (applicable only on specific system) and the system-independent (applicable on any systems) [74]. The former may be more efficient, however they are limited, while the latter can be applied to any systems as a complement to an existing system-specific solution. Examples of system-independent load management solutions for file search are caching [6] and paging [58]. A cache is a high-speed "memory" that acts to minimize the number of accesses to a large low-speed "memory". For example, in web systems, the high-speed "memory" could be hard disk of a computer acting to cache web pages fetched from the web (i.e., low-speed "memory"). While paging is often used for displaying a large amount of search results in alleviating server-client traffic load. For instance, if a result set is less than 50 rows, paging may be optional. However, if it is bigger than 5×10^4 rows, paging is practically required for splitting the result set. Caching and paging have been widely adopted in many systems [33, 93, 45, 23, 92].

Although such general techniques used in web systems have been adopted into P2P systems, it is highly insufficient and some of system-specific solutions are strongly required for time-consuming search operations. Novak [56] analyzed the source of the load related to response time. They suggest that the load should be measured as two separate quantities, i.e.,

- **processing load** is taken as an average time of processing a search query on the local data of a given peer. This value considers the *data volume*, the complexity of the queries, and quality of the local index.
- **waiting load** is measured as an average time a query is waiting in the queue before processed. This value considers the *frequency of queries* and the **processing load** of particular peers.

Mondal *et al.* [52] analyzed the best selection between the options of data migration and data replication in a cluster based P2P networks, and formulated a strategy based on run-time decision described as follows.

- "Pushing" non-hot data (via migration for large sized data and via replication for small-sized data) to large capacity peers as much as possible.
- Replicating small-sized hot data at small capacity peers (smaller search space expedites search operations).
- Large-sized hot data are migrated to large capacity peers only if such peers have low probability of leaving the system, otherwise they are replicated at large capacity peers, an upper-limit being placed on the number of replicas to save disk space.

However, their strategy is very abstract and the management cost of object migration and replication is very high in the system.

In Chapter 3 and 4, we focused on two kind of super-peer based *hierarchical architectures* of P2P systems, which are distinguished by the organization of super-peers. In each of them, we discussed system load allocation, and proposed index migration or replication based load balancing algorithms for alleviating load imbalance of super-peers, aiming to decrease average and variation of query response time during index retrieval process.

Chapter 3

Load Balancing for Qin's Hierarchical P2P system

3.1 Introduction

In this Chapter, we focus on a hierarchical P2P proposed by Qin *et al.* [61]. This is a three-tier P2P architecture consisting of top, middle, and bottom layers, where indices of files held by the **user peers** (UPs, for short) in the bottom layer are stored at the **super-peers** (SPs, for short) in the middle layer, and the correlation between those two layers is maintained by the central server(s) in the top layer using the notion of tags (see Section 3.3.1 for the details). As will be described later, in this system, an overloaded SP easily becomes a bottleneck of query processing, which would significantly degrade the overall performance of information retrieval. In other words, Qin's hierarchical P2P has a great potential for striking a good trade-off point between the search efficiency and the system cost if we could remove such bottleneck by *balancing* the load of SPs, although it was not explicitly addressed in [61].

In this Chapter, we address several issues in P2P load balancing. In Qin's system, the **task** of an SP is defined by a collection of indices held by the UPs associated with the SP, and is represented by a set of tags associated with the collected indices. Although the load imbalance of SPs can be alleviated by migrating excessive tasks from heavily loaded SPs to lightly loaded SPs as in many previous load balancing schemes [67][16], such a simple approach is not sufficient if the size of tasks is highly imbalanced, i.e., if the preference of users concerned with tags follows a power-distribution such as the Zipf's law [55]. To overcome such a critical issue, we will propose two load balancing schemes based on the *resizing* of tasks. The first scheme tries to bound the maximum size of tasks by splitting large tasks to have a size not exceeding a threshold μ. A mechanism which efficiently realizes such a task splitting with low overhead is also proposed. The second scheme controls the size of each task by updating the "definition" of tasks. More concretely, we conduct a reordering of the priority of tags used in the query forwarding step in Qin's system (see Section 4.4 for the details). The performance of the proposed schemes is evaluated by simulation. The result of simulations indicates that the proposed schemes effectively reduce the maximum load, and ensure an even load distribution over SPs.

The remainder of this Chapter is organized as follows. Section 3.2 summarizes the characteristics of Qin's system. Section 3.3 overviews Qin's hierarchical P2P. Section 3.4 proposes two load balancing schemes. The result of evaluation is given in Section 3.5. Finally, Section 3.6 concludes this Chapter.

3.2 Observation of Qin's Hierarchical P2P Architecture

In this model, the bottleneck of the index server used in Napster [54] could be certainly overcome by replacing the single index server by a collection of super-peers, and the low search efficiency of the fully decentralized P2Ps could be overcome by introducing top and middle layers (i.e., the central server layer and the super-peer layer). In other words, this hierarchical model has a great potential for striking a good trade-off point between the search efficiency and the system cost if the load of super-peers is well balanced.

There are two attractive factors of this architecture. The first factor we have to note here is that by adopting this architecture, the load of overall system is optimally distributed among three layers. More concretely,

- The load of the central server is sufficiently low, since it does not conduct a search of files relevant to a given query, i.e., it merely forwards a received query to super-peers.
- The message transmission over the network is effectively controlled by the central server, since a query received from a user peer is merely forwarded to those super-peers relevant to the query, i.e., it effectively avoids flooding of messages.
- The load imbalance of super-peers can be significantly decreased by adopting load balancing schemes (see Section 3.4 for the details) , i.e., it guarantees a quick response to a given query.

The second factor we concerned is the real-time search of shared files. By combining an event detection mechanism at each user peer with a collection mechanism at the super-peers, this architecture can realize a real-time update of file indices collected from user peers. In addition, this architecture allows a user peer to send a query to both of central server and its corresponding super-peers. After receiving a forwarded query from the central server or a query directly received from its local user peers, the super-peer will process the query and return search results to the requester as an independent search engine.

3.3 System Model

In this section, we describe an outline of hierarchical P2P proposed by Qin *et al.* which is aiming to realize a real-time file search in distributed networks. Moreover, we describe a skewed query model assumed in this Chapter.

3.3.1 Design

Consider a hierarchical P2P consisting of three layers, where the top layer consists of a central server S_0, the middle layer consists of a set of super-peers (SPs) $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, and the bottom layer consists of a number of user peers (UPs) [61]. Figure 3.1 illustrates an overview of the P2P architecture.

In this system, each file held by a UP is associated with an SP so that an index of the file is maintained by its corresponding SP. Such a mapping of files is controlled by a set of "tags" attached to each file. Let T be the set of all tags attached to the files. Let σ be a bijection from T to $\{1, 2, \ldots, |T|\}$, where in the following, $\sigma(t)$ is called the **priority** of tag t under σ. Given tag set $T' \subseteq T$, the priority sequence of T' under σ is defined as a sequence of elements in T' which is arranged in the increasing order of their priority under σ. Given two tag sets $T_1, T_2 \subseteq T$, T_1 is said to be **included** by T_2 under σ, if the priority sequence of T_2 is a prefix of the priority sequence of T_1.

By using such an inclusion relation between tag sets, we can define a mapping of files to SPs in the following manner:

1. Each SP S_i is associated with a tag set T_i, where concrete procedure to determine a set of tag sets assigned to SPs will be described later.

2. Each file x is attached a tag set $T(x)$ by its content holder.

3. File x is mapped to S_i if T_i is a minimal tag set including $T(x)$, where T_i is said to be minimal if no tag sets in the current system are included by T_i (the

reader should note that a minimal tag set has a longest priority sequence).

The set of "tag sets" which will be assigned to SPs is determined by using a tree structure defined as follows (in what follows, we call it a **discrimination tree**):

1) The root of the tree is associated with an empty set. Tag sets associated with the other vertices are determined by using the following procedure recursively.

2) Let $T(u)$ denote a tag set associated with vertex u. Let t' be a lowest priority tag in $T(u)$ and let $i' = \sigma(t')$ for brevity.

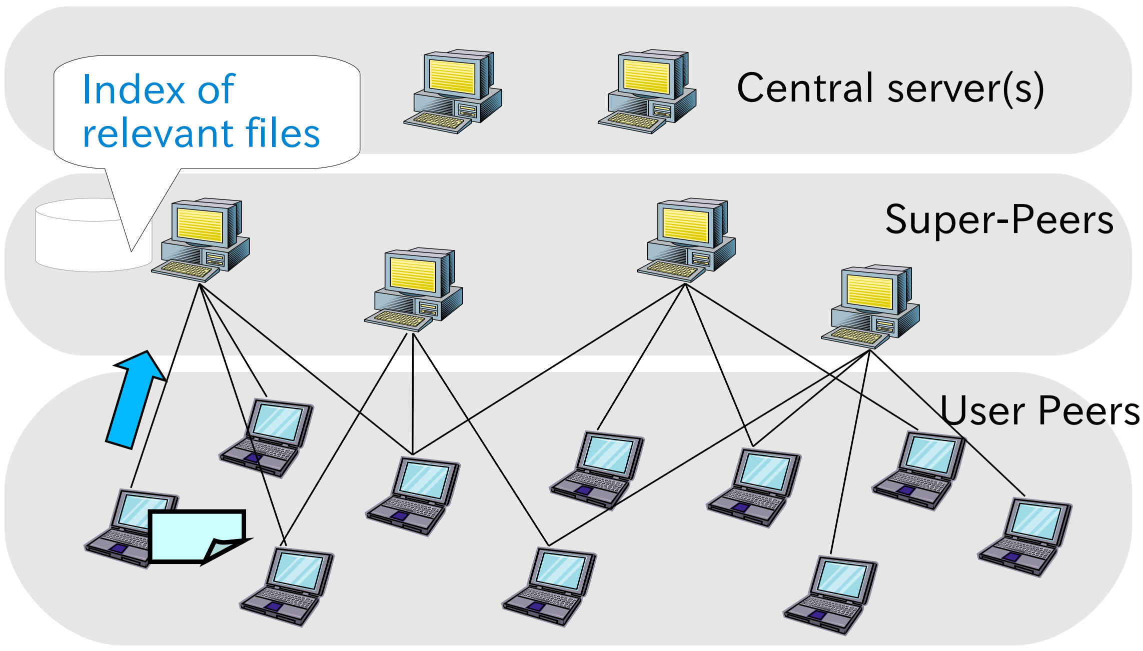
3) Then, vertex u has at most $|T| - i'$ children each of which is associated with tag set $T(u) \cup \{\sigma^{-1}(j)\}$ for $i' + 1 \le j \le |T|$.

4) If the number of children of u is smaller than $|T| - i'$, we may select any tag set from the candidates, and attach them to the children.

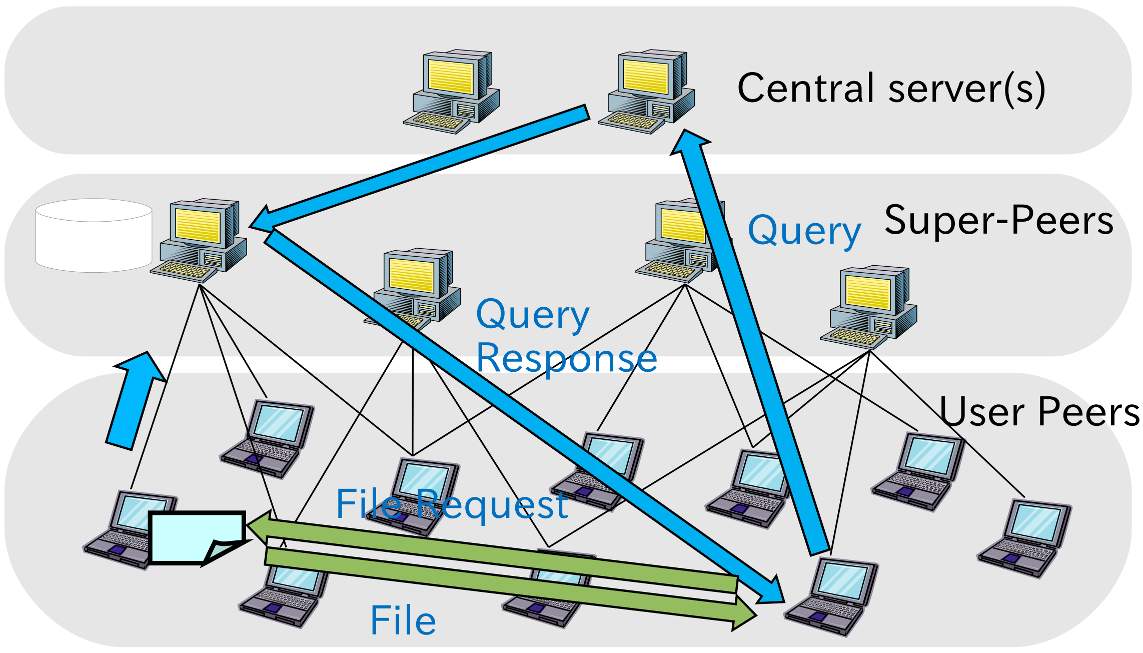
The reader should note that the above procedure realizes the split of a task into several subtasks, as will be described in Section 4.3. In the Qin's system, such a tree structure (i.e., discrimination tree) is used to find a tag set matching a given query, in such a way that the query is initially placed at the root vertex and moves toward a vertex associated with a minimal tag set including the query (note that the tree is constructed so that any tag set on the path connecting to the minimal tag set includes the given query).

3.3.2 Skewed Query Model

Suppose that the system receives a query Q consisting of several tags. After receiving it, the system retrieves the network to return a set of files attached all tags contained in Q. In this Chapter, we assume that there are no correlation between tags. Let p_i denote the probability of associating tag $t_i \in T$ to a query (or a file), and without



(a) File Upload procedure.



(b) File Retrieval procedure.

Figure 3.1: A model of three-tier P2P architecture.

loss of generality, we assume that $p_i \geq p_{i+1}$ for any $1 \leq i \leq |T| - 1$. More specifically, we assume that it follows the Zipf's first law, i.e.,

$$p_i = \frac{1}{i^s \times H_{|T|,s}}, \tag{3.1}$$

where

$$H_{|T|,s} \overset{\text{def}}{=} \sum_{1 \leq i \leq |T|} (1/i^s) \tag{3.2}$$

and s is a parameter called Zipf-parameter. Note that $\sum_{i=1}^{|T|} p_i = 1$ by definition. The length of each query is assumed to follow a binomial distribution as is roughly verified using actual statistical data [85]. More concretely, we assume that each query is constructed using a Bernoulli trial with stopping probability p; i.e., the probability that a given query has length i is given by as follows:

$$|Q| \overset{\text{def}}{=} (1-p)^{i-1} \times p. \tag{3.3}$$

It should be worth noting that under such natural and reasonable assumptions, the number of queries associated with each tag will be highly imbalanced; i.e., a popular tag is contained in a large number of queries, while an unpopular tag is rarely contained in the queries. In other words, an SP which is associated with a popular tag should receive a significant number of queries which causes an overload of the SP.

3.4 Task Migration based Load Balancing Schemes

In this section, we first describe a formal definition of the load balancing problem. After illustrating a simple task migration scheme for solving the problem, we propose two concrete load balancing schemes, i.e., 1) a task splitting scheme to ensure that the size of each task is bounded by an appropriate value, and 2) a tag reordering scheme to bound the depth of the discrimination tree used in the query forwarding step.

3.4.1 Preliminaries

In the following, we assume that each SP is associated with several tag sets corresponding to tasks, and will use words "task" and "tag set" interchangeably. Let $D = \{d_1, d_2, \ldots, d_{|D|}\}$ be a set of tasks in the system. Let $w(d_i)$ denote the number of files associated with d_i, and let $v(d_i)$ denote the number of queries included by d_i.

In this Chapter, to reflect both of the number of files indices (i.e., spatial cost) and their access frequency (i.e., temporal cost), we define the **load** of task d_i as follows:

$$\ell(d_i) \stackrel{\text{def}}{=} w(d_i) \times v(d_i). \tag{3.4}$$

A mapping of tasks in D to the set of SPs in $\mathcal{S}$ is represented by a binary matrix C such that $C[i,j] = 1$ iff d_i is assigned to S_j. The **load** of S_j, denoted as $\ell(S_j)$, is the summation of the load of tasks assigned to it, i.e.,

$$\ell(S_j) \stackrel{\text{def}}{=} \sum_{1 \le i \le |D|} \ell(d_i) \times C[i,j]. \tag{3.5}$$

The load balancing problem is now stated as follows:

Definition 3.4.1 (Load balancing problem) Find matrix C to minimize $\max_{1 \le j \le m} \ell(S_j)$.

In this Chapter, we propose heuristic schemes to solve the above load balancing problem in P2P environment. In describing the details of the proposed algorithm, we use several notations as follows: Let b_i be a Boolean variable which represents whether or not the load of S_i exceeds a threshold θ. Threshold θ is referred to as the target load of SPs which is typically defined as follows:

$$\theta \stackrel{\text{def}}{=} \frac{\sum_{d_i \in D} w(d_i) \times \sum_{d_i \in D} v(d_i)}{|D| \times |\mathcal{S}|} + c \tag{3.6}$$

where $c > 0$ is an appropriate slack. The load of S_i is said to be **heavy** if $b_i = \mathsf{true}$, and **non-heavy** otherwise. Let $B = (b_1, b_2, \ldots, b_n)$ be a Boolean vector representing the load of SPs, where in the following, we assume that vector B is maintained by central server S_0 in its local storage. Finally, we should remind that in our hierarchical P2P architecture, the migration of tasks is realized by modifying binary matrix C, and by transferring a set of indices corresponding to the migrated tag sets to their destination.

3.4.2 Task Migration Protocol

In this subsection, we describe a protocol to migrate excessive tasks of heavy SPs to non-heavy SPs, which will be used as a basic procedure in the proposed schemes.

Let S^* be a heavy SP, and D^* be the set of tasks assigned to S^*. By definition, $\ell(S^*) > \theta$. At first, S^* notifies S_0 that it becomes heavy. Upon receiving the notification message, S_0 updates vector B to reflect the change of the status, and returns S^* a list of non-heavy SPs which can accommodate excessive tasks of S^*. After receiving the list, S^* tries to hand over (a portion of) excessive tasks to the recommended SPs until it satisfies $\ell(S^*) \leq \theta$ or the list of non-heavy SPs becomes empty.

Task migration between two SPs is executed as follows. Let $\hat{D}$ be the set of excessive tasks of S^*, and suppose that S^* selects SP S' as the target of task migration. Before starting a task migration, S^* acquires the available capacity δ ($\overset{\text{def}}{=} \theta - \ell(S')$) of S' by directly communicating with S'. It then determines the set of tasks to be migrated to S' as follows:

```
function TASK_SELECTION($\delta$)
Step 1: Let $Q = \emptyset$.
Step 2: If $\delta < \min_{d \in \hat{D}} \ell(d)$, then go to Step 4.
Step 3: Let $d'$ be a task with largest load in $\hat{D}$ such that $\ell(d') \leq \delta$. It moves $d'$ from
    $\hat{D}$ to $Q$, decrements $\delta$ by $\ell(d')$, and go to Step 2.
```

Step 4: Output Q, and terminate.

After that, S^* sends a copy of index set associated with tasks contained in Q to S', and after completing such an index transfer, it sends a message to S_0 to change the mapping of tasks to S^* and S'.

3.4.3 Task Splitting

In the task migration protocol described above, we could not avoid a situation in which no tasks in a heavy SP can be migrated to other SPs if the load of tasks is highly imbalanced. For example, if the target load θ is set to 10, and if S_1 is associated with three tasks of load $10, 15$, and 20, respectively, then an excessive task with load 15 or 20 cannot be migrated to any other SPs under the protocol.

The key idea of our first scheme is to bound the maximum load of tasks by an appropriate constant μ satisfying $\mu \leq \theta$. In the following, the load of task d_i is said to be heavy if $\ell(d_i) > \mu$. Suppose that SP S^* is assigned a heavy task d^*. In the proposed scheme, after recognizing the heaviness of task d^*, S^* splits d^* into several subtasks according to a given bijection σ, in the following manner:

function TASK_SPLITTING(d^*)

Step 1: Let $P := \emptyset$.

Step 2: Consider a tag set associated with d^*. Let t' be a lowest priority tag in the tag set, and $i' := \sigma(t')$ be the priority of t' in σ.

Step 3: Recall that d^* can have at most $|T| - i'$ children in the discrimination tree. For $i := i' + 1$ to $|T|$, repeat the following procedure until $\ell(d^*) \leq \mu$: 1) add a new child to d^*; 2) associate tag set $d^* \cup \{\sigma^{-1}(i)\}$ to the new child; 3) move the load of d^* concerned with the tag set to the new child; and 4) add the new tag set to P.

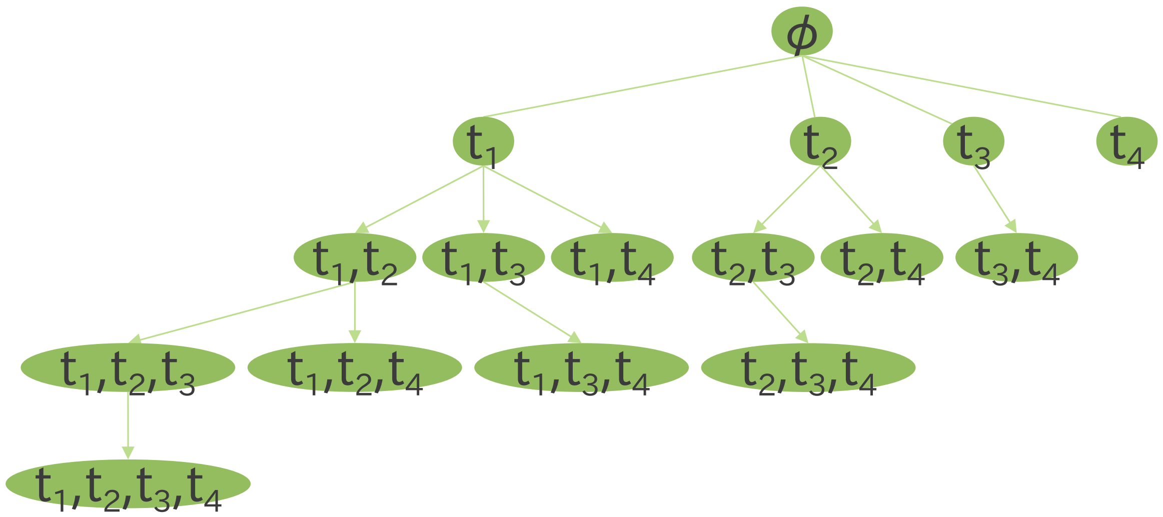


Figure 3.2: An example of task splitting when $|T| = 4$.

Step 4: Output P, and terminate.

Figure 3.2 illustrates task splitting when $|T| = 4$. The number of possible tasks consisting of k tags is given by $C(|T|, k) = \frac{|T|!}{(|T|-k)!k!}$, and the number of all possible tasks over T is $2^{|T|} - 1$. After completing the task splitting, it applies the task migration scheme to balance the load of SPs.

Task integration is a reverse operation of task splitting, which is effective to reduce the depth of the discrimination tree and the number of tasks manipulated by the central server. Recall that the depth of the tree corresponds to the response time of query forwarding and the size of the tree corresponds to the overhead of each operation conducted by the central server. Basically, a task integration is simply realized by removing a task from the discrimination tree since the load of a removed task should always be migrated to its parent. Thus, if we use such an integration operation, we should remind that the load of the parent does not exceed the target load, while it is left as a future work.

3.4.4 Tag Reordering

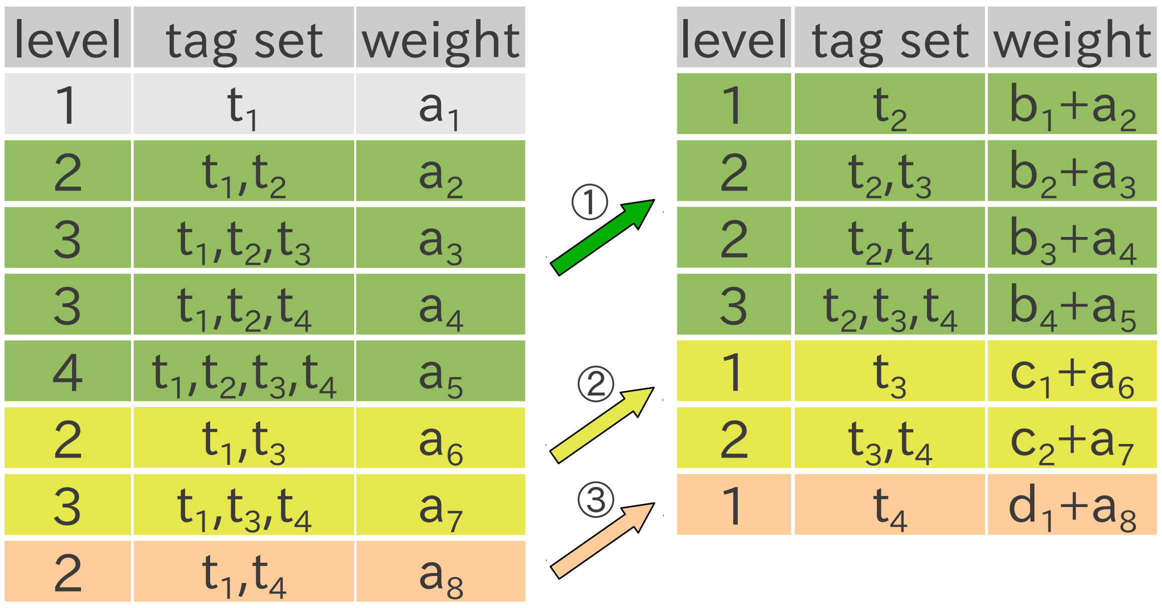
Under the skewed query model introduced in Section 3.2, each query consists of several tags, and the length of most queries is bounded by a constant L. This means that a tag set larger than L will rarely match the queries, which would cause a significant load imbalance even if the number of files associated with each task was evenly balanced. On the other hand, it is also true that under the first scheme, a tag set containing many popular tags must be split into a number of smaller subsets so that the size of each subset is sufficiently small.

The objective of our second scheme is to resolve such a conflict by conducting a reordering of tags in the given bijection σ. More concretely, it updates σ so that popular and high priority tags are moved to the end of the tag sequence derived from σ. Let d^* be a heavy task assigned to SP S^*. Suppose that tag set d^* contains more than L tags, and that t^* is the lowest priority tag in d^*. Upon detecting the heaviness of such task d^*, S^* asks S_0 to reduce the priority of t^* in the set T of all tags. After receiving such an inquiry, S_0 conducts a tag reordering in the following manner:

function TAG_REORDERING(t^*)

Step 1: Let $i' := \sigma(t^*)$. For each $(i'+1) \leq i \leq |T|$, S_0 conducts the following operations: 1) $t_i := \sigma^{-1}(i)$; and 2) $\sigma(t_i) := i - 1$. It then updates $\sigma(t^*)$ such that $\sigma(t^*) := |T|$.

Step 2: After completing the update, S_0 broadcasts a message so that tag t^* is removed from all tag sets associated with SPs. If a tag set associated with a vertex in the discrimination tree becomes identical to the tag set associated with its parent after the removal, it simply removes the child vertex from the tree, and migrates file indices associated to the child to the SP associated to the parent vertex.



level	tag set	weight
1	t_1	a_1
2	t_1,t_2	a_2
3	t_1,t_2,t_3	a_3
3	t_1,t_2,t_4	a_4
4	t_1,t_2,t_3,t_4	a_5
2	t_1,t_3	a_6
3	t_1,t_3,t_4	a_7
2	t_1,t_4	a_8

level	tag set	weight
1	t_2	b_1+a_2
2	t_2,t_3	b_2+a_3
2	t_2,t_4	b_3+a_4
3	t_2,t_3,t_4	b_4+a_5
1	t_3	c_1+a_6
2	t_3,t_4	c_2+a_7
1	t_4	d_1+a_8

Figure 3.3: An example of task merge when $|T| = 4$.

Figure 3.3 illustrates tag reordering when $|T| = 4$. After reordering the priority sequence from $1, 2, 3, 4$ to $2, 3, 4, 1$, a number of tasks will merge into existing tasks.

3.5 Simulation Model

We evaluate the performance of the proposed load balancing schemes by simulation. The simulator is written in Java, and is developed under the following environment: open-SUSE/10.1, Intel Core$^{\text{TM}}$ 2 Duo CPU 3.00GHz, Memory 2GB, Eclipse/3.4, JDK/1.6.

In the set of simulations, we omit a concentration of task migration to a non-heavy SP which may cause a chain effect of task migration; i.e., the non-heavy SP becomes next heavy SP and continue to migrate its tasks to the other SPs. Note that such a situation can be avoided using an appropriate lock mechanism during task migration in a real system.

3.5.1 Simulation Model

In the following, we fix $|\mathcal{S}| = 100$ and $|T| = 500$, i.e., we do not consider join and leave of SPs. In the beginning, each SP is associated with 5 non-overlapped tags. The probability of associating tags to a query or a file follows the same Zipf's distribution with parameter $s = 1.0$. The length of each query is determined by a Bernoulli trial with stopping probability 0.4, i.e., the probability of assigning i tags is given by $(0.6)^{i-1} \times 0.4$, while the maximum length of each query is bounded by 10.

In many social tagging systems such as Delicious and Yahoo! Bookmarks, most users attach 2 to 5 tags to each file. In this Chapter, we assume that the number of tags attached to each file follows a normal distribution with mean 10 and standard deviation 2.5 (i.e., a large portion of files are assigned around 10 tags), while it is bounded by 20. The total number of queries and files are both fixed to 10000. Finally, bijection σ is initially given at random.

3.5.2 Effect of Load Balancing

We first evaluate the performance of three load balancing schemes. Figure 3.4 summaries the result. The vertical axis is the load of each SP and the horizontal axis is SPs arranged in an ascending order of the load. Four curves in the figure show the initial distribution of the load and the load distributions after applying three schemes, respectively. In order to obtain good load distribution of SPs, we set a small target load value $\theta = 2100$ in all simulations. Here, it is worth noting that how to set an appropriate threshold value of task size μ is also important. If μ is set to a high value, the load over tasks may be still highly imbalanced, and if μ is set with a low value, it may accelerate task splitting. Empirically, the maximum load of tasks used in the task splitting scheme is set as $\mu = 300$.

As shown in Figure 3.4, although it could reduce the ratio of heavy SPs from 22%

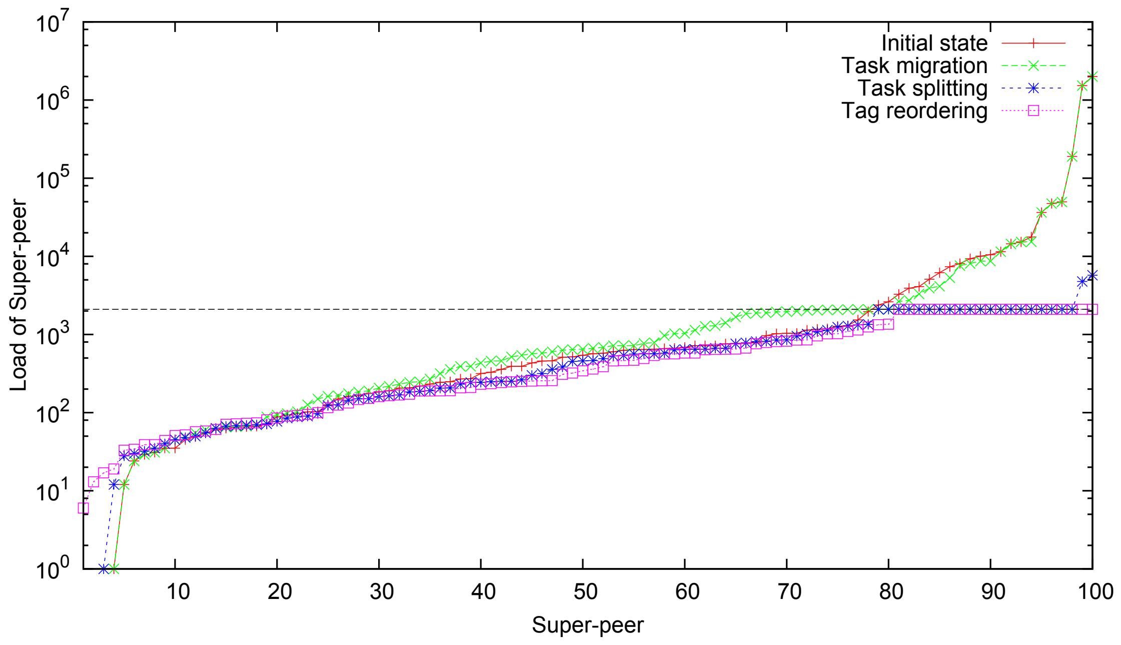


Figure 3.4: Effect of load balancing scheme.

to 20%, the effect of the task migration scheme is very limited. In contrast, the ratio of heavy SPs could be significantly reduced by applying proposed methods. More concretely, the ratio of heavy SPs reduces to 2% by the task splitting scheme, and it further reduces to zero by the tag reordering scheme. In other words, we could attain a sufficient load balancing by our first scheme, while there are few cases which could not be resolved by the first scheme but could be effectively resolved by our second scheme.

Table 3.1 compares the cost of migration based load balancing schemes in terms of the percentage of indices exchanged among SPs. This result indicates that our two proposed schemes realize an even load balancing by spending more cost than the simple task migration scheme. In addition, we could observe a remarkable feature of the schemes such that the total cost of task splitting significantly reduces by applying the tag reordering; i.e., an inappropriate ordering of tags causes a number of splitting

Table 3.1: Cost of migration

Load balancing	Percentage of Moved indices
Initial State	NULL
Task Migration	5.24%
Task Splitting	43.65%
Tag Reordering	18.85%

of tasks to realize a fair distribution of the load, which could effectively be removed by conducting a tag reordering.

3.5.3 Spatial Load vs. Temporal Load

In the proposed schemes, the load of SPs is defined as a product of spatial and temporal metrics as was described in Section 4.1. In this subsection, we examine how each of those metrics could be balanced under the proposed load balancing schemes.

Figure 3.5 shows the distribution of the number of indices held by each SP (i.e., spatial load) under three load balancing schemes. We can observe from the figure that by using proposed schemes, a high concentration of indices assigned to three SPs is effectively distributed to 25 SPs, whereas the simple task migration scheme cannot migrate excessive tasks due to the imbalance of the load of tasks. In fact, the standard deviation of the number of indices assigned to SPs is summarized as follows: 1) before conducting a load balancing, the standard deviation is 284.82, 2) after applying the task migration scheme, it slightly reduces to 282.30, and 3) by applying task splitting and tag reordering schemes, the standard deviation significantly reduces to 148.82 and 146.08, respectively.

Figure 3.6 shows the distribution of number of queries (i.e., temporal load) under three load balancing schemes. We can make a similar observation to the case of spatial

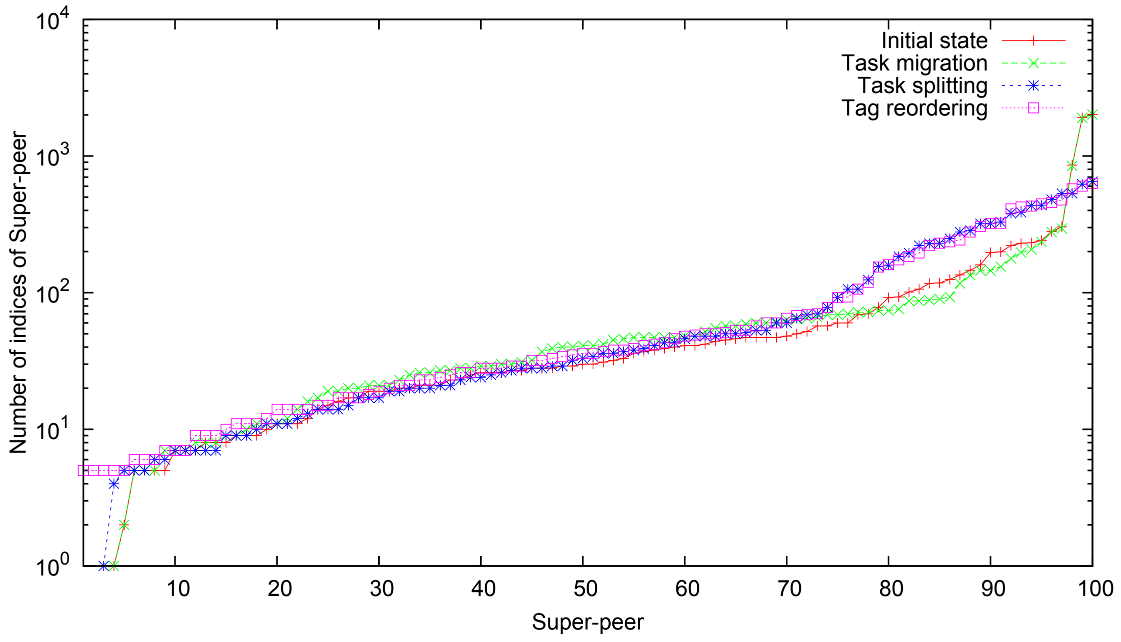


Figure 3.5: Distribution of spatial load.

distribution, where the standard deviation of the schemes is summarized as follows: 1) before conducting a load balancing, the standard deviation is 320.49, 2) after applying the task migration scheme, it slightly reduces to 305.17, and 3) by applying task splitting and tag reordering schemes, the standard deviation significantly reduces to 150.98 and 149.97, respectively.

3.6 Summary and Discussion

In this Chapter, we studied the load balancing problem in Qin's hierarchical P2P architecture, and proposed two load-balancing schemes based on the notions of task splitting and tag reordering. Those schemes help to avoid overloading super-peers for this system. We evaluated the performance of the schemes by simulation, and the results demonstrate the effectiveness of these schemes.

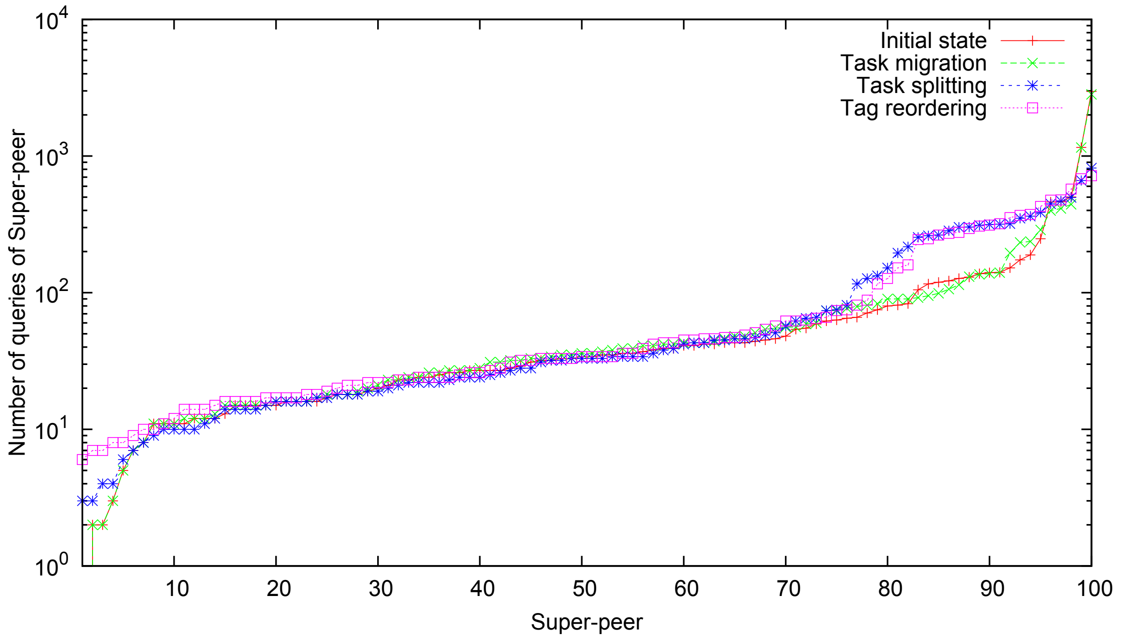


Figure 3.6: Distribution of temporal load.

We have some thoughts about the system design and migration schemes. Firstly, richness of search result for a given query could be improved. In system design (See Section 3.3.1 for the details), a given query is only match to the task with the longest prefix (i.e., destination task). Such a restriction on search algorithm can be easily relaxed. For example, parent and child of destination task in the discrimination tree could be the candidate of providing search results. Secondly, if we relax the search algorithm, a given query would be forwarded to several super-peers if the corresponding destination tasks of the query is maintained at different super-peers. Therefore, a middleware at user-peers to receive parallel computed search results needs to be implemented. Thirdly, the performance of load balancing schemes such as scalability and dependability needs to be tested in a more realistic P2P environment such as heterogeneous bandwidth of super-peers, and join and leave of super-peers.

Chapter 4

Load Balancing for DHT-based Super-Peer Systems

4.1 Introduction

In this Chapter, we focus on a two-tier hierarchical P2P [25]. In this architecture, indices of files held by the **user peers** in the lower layer are stored at the **super-peers** in the upper layer. The same user peers could be grouped into different super-peers according to a certain policy (e.g., the proximity of their geographical locations), and each group is required to have one super-peer. The super-peers are gateways between the groups, and they are connected based on DHTs.

To retrieve indices, multi-keyword search is a quite common type, and it can be realized by using the solution for single keyword (or key) search with an intersection or union operation of indices. More concretely, by given a multi-keyword query initiated by a user, it will be parsed into multiple sub-queries, each of which corresponds to a keyword contained in the query. Those sub-queries are sent to corresponding super-peers by using the DHT. The search result received by the user is an intersection or union of the indices obtained by each sub-queries. In this process, since the preference of users concerned with keywords contained in queries generally follows a power-law distribution such as Zipf's law [55], an access concentration of queries would probably happen at two kinds of super-peers, i.e., i) home super-peers that hold indices associated with popular keywords, and ii) intermediate super-peers who are in the intersection of multiple routing paths towards "hot" home super-peers. In the system, those two kinds of super-peers are easily become a bottleneck of query processing. If there are several such overloaded super-peers in the routing path for a given query, it would incur long index retrieval latency and consequently negatively influence the overall performance of object retrieval.

Index replication has a great potential for alleviating this issue. However, existing schemes did not explicitly address it or incurred high cost [65][68][75]. Management cost of index replicas (i.e., cost of copying and keeping index replicas) is very small. For example, suppose that an entry is approximately 500B, an index consists of 200 entries would be just approximately 100KB. On the other hand, maintenance cost of index replicas may be high if we want to guarantee the consistency between an index and its replicas, since the index entries associated to a keyword would be continuously inserted or deleted. The degree of overhead for guaranteeing index consistency may be varied according to the need of real systems, and it is an open problem in this Chapter. Generally, the cost of index replication is evaluated through the number of replicas rather than the size of replicas, i.e., the maintenance cost is assumed far larger than the management cost. Instead of index replication, pointer would be an effective approach to avoiding the consistency issue of index replication, where a pointer is a pair of key (i.e., hashed keyword) and index location. In other words, the pointer is a "shortcut" between a search key and a home super-peer of the keyword.

Replica placement has a significant impact on minimizing the replication cost. The replica placement problem concerned on optimizing one kind of resources in P2P systems is found to be NP Complete [102], i.e., optimal placement of replicas will always lead to exponential time algorithms. Alternatively, a lot of heuristic algorithms have been proposed to give approximate solutions for the problem. Majority of existing replication methods place replicas in the query paths to intermediate super-peers between requesters and home super-peers, referred to as path-based replication or probabilistic replication. PCache [68] and EAD [75] are two prominent probabilistic replica placement schemes. On the other hand, deterministic replication [28] has also been proposed based on DHTs. In deterministic replication, each point in a hash space is associated with a set of other identified points, i.e., if a point i is associated with identified point r, then any (key, value) pair with point i should be stored at the super-peers responsible for points i and r. To realize benefits of these approaches for minimizing the replication cost, it is necessary to instantiate particular schemes for different classes of DHTs, utilizing their routing characteristics.

In this Chapter, we focus on a class of tree-based routing DHTs (See Section 4.3.1 for the details). In such DHT-based super-peer systems, we will propose an integrated solution consisting of three replication schemes to alleviate load imbalance of super-peers while minimizing the cost. The first scheme tries to bound the maximum response load received by each index to have it not exceeding a threshold μ. Under such constraint, an index with a popular keyword (i.e., popular index) will make more replicas, so that routing load for the popular index would be highly reduced and response load received by the popular index can be shared by its home super-peer and replica super-peers. The second scheme places a predetermined number of pointers for each index, aiming to decrease the number of index replicas generated by the first scheme. The third scheme controls the maximum query load of super-peers by operations of pushing or pulling index replicas, in order to eliminate bottleneck situation of query processing in the system. The result of simulations indicates that all the proposed schemes can work in coordinate, in alleviating the query load imbalance of super-peers. Moreover, by comparison it was proved that our schemes are more cost-effective on placing replicas than PCache and EAD.

The remainder of this Chapter is organized as follows. Section 4.2 summarizes the characteristics of DHT-based super-peer systems. Section 4.3 describes the details of system model. Section 4.4 proposes three replication based query load balancing schemes. The result of evaluation is given in Section 4.5. Finally, Section 4.6 concludes this Chapter.

4.2 Observation of DHT-based Super-Peer Architecture

Since super-peers act as centralized servers to the user peers, they can handle queries efficiently. On the other hand, since there are relatively many super-peers in this architecture, there is no single super-peer has to handle a very large load, avoiding a bottleneck and/or single point of failure for this architecture. In order to optimally allocate system load in this architecture, it is well-known that gossip mechanism can be used to construct super-peer layer in the following way [53], i.e., a user peer may decide to become a super-peer and take responsibility for some of the clients of the other peer, to allocate more system load in super-peer layer; alternatively, a super-peer may decide to move all its user peers to the other super-peers and become a user peer, to reduce the number of super-peers and thus reduce the load generated by query routing between super-peers.

In this architecture, the network performance can be measured based on two types of load metrics [98], i.e., a) individual load (the load of a single peer), and b) aggregate load (the sum of the loads of all peers in the system). Increasing the number of super-peers in the network, it will reduce the load of the individual super-peers. However, it increases the aggregate load in the network since the routing load in the overlay is increased, hence it is necessary to find the balance in this architecture.

Therefore, load management for DHT-based super-peer architecture includes two aspects, i.e., 1) reduce the aggregate load of super-peers, and 2) balance the load of super-peers.

4.3 System Model

In this section, we describe a basic keyword search model based on a class of tree-based routing DHTs. Moreover, we describe a skewed query model assumed in this Chapter.

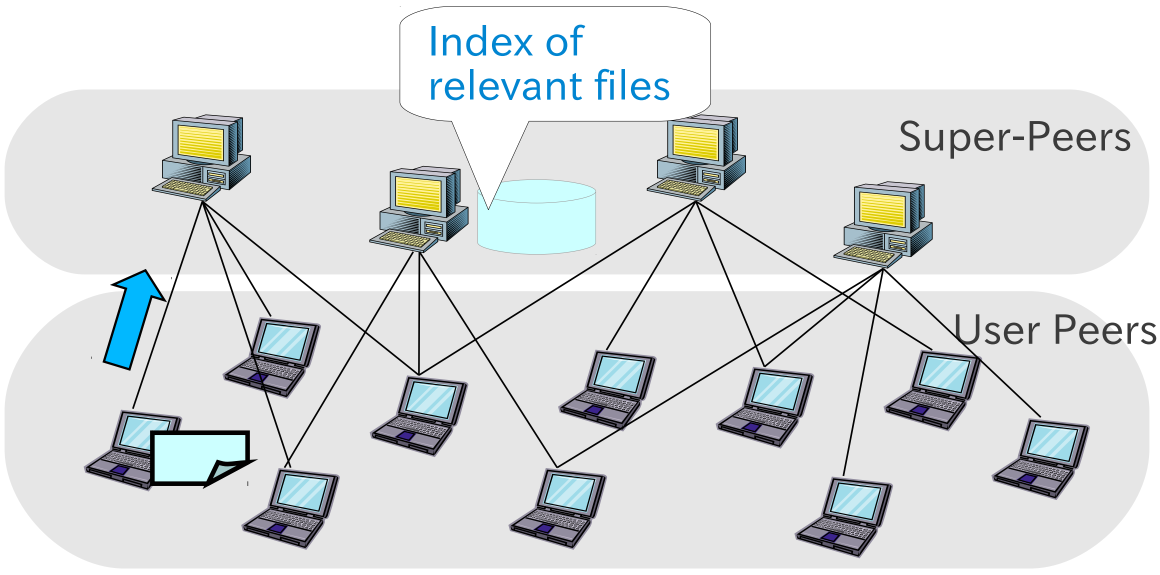
4.3.1 Design

Consider a two-tier super-peer system, where the upper layer consists of a set of DHT-based super-peers (SPs) $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, and the lower layer consists of a number of user peers (UPs) [25]. Figure 4.1 illustrates an overview of DHT-based super-peer architecture.

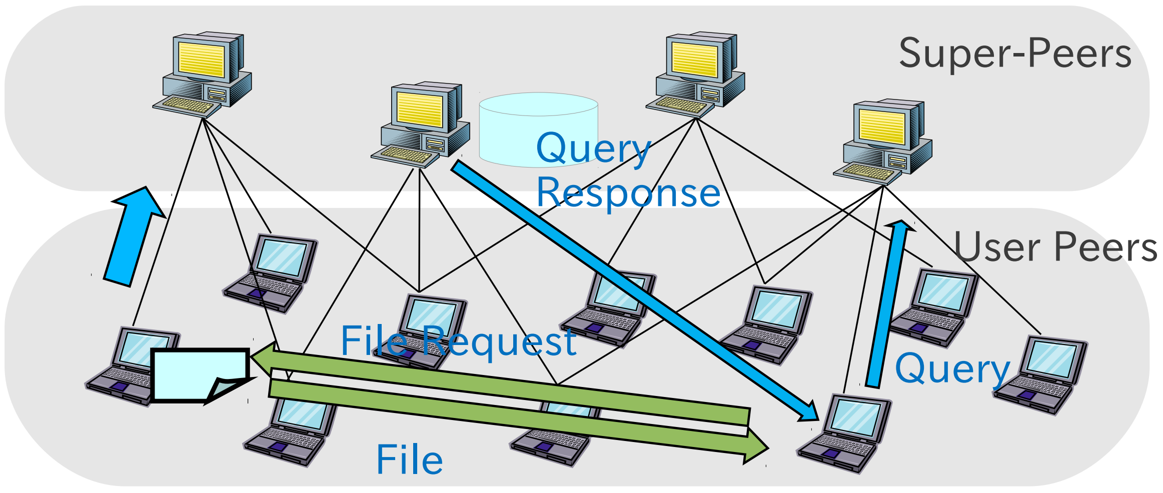
We consider a class of tree-based routing DHTs defined in [35]. In such DHTs, paths from any requester peer to all possible home peers that stored indices are aggregated, the resulting topology is a tree. Many popular DHTs belong to this category such as Chord [82], Pastry [73] and Tapestry [101].

More concretely, in this Chapter, we adopt Chord (an instance of tree-based routing DHTs) to super-peer layer. In the system, index entry of each object held by a UP is maintained by its corresponding SPs. Such a mapping is controlled by a set of keywords attached to the object. By using consistent hashing and routing protocol of Chord, we can define a mapping of objects to its corresponding SPs in the following manner:

1. Each object x is attached a keyword set $D(x)$ by its content holder.



(a) File Upload procedure.



(b) File Retrieval procedure.

Figure 4.1: A model of DHT-based super-peer system.

2. For each $d_i \in D(x)$, object x is mapped to S_i if S_i is a successor of hashed d_i.

For example, given keywords d_1, d_2, d_3 assigned to an object f_0, and SPs of d_1, d_2, d_3 are S_1, S_2, S_3, respectively. Index entry of f_0 will be mapped to these three SPs. This ensures indices of objects to be approximately[1] evenly distributed over participating SPs. An example of an index entry is given in Table 4.1.

Table 4.1: Structure of an index entry

Key	Metadata	Location
hash(d_1)	album, artist, year	user peer ID: A

As shown in Table 4.1, an index entry consists of three main fields, i.e., key, metadata and location. The metadata field is used to help users choosing objects to download, as well as supporting metadata search in the system. Here, it is worth noting that an index is a set of entries associated to a given keyword, and an index replica is a copy of the whole entries contained in the index.

Conducting a search for a single keyword is to reveal all of the index entries containing the keyword. The search key is mapped to a SP using Chord. The index for the computed key will be delivered to the requester. To answer a boolean "AND" or "OR" query that contains multiple keywords, it fetches the indices of each keyword and calculate the intersection or union of obtained indices as the final result. In the following, we consider the basic keyword search model as described above. But, it is worth noting that by carefully caching the search result of frequently requested keyword sets and parsing queries into proper phases, the performance of keyword search could be improved.

[1]Consistent hashing produces a bound of $\mathrm{O}(\log n)$ imbalance degree of keys between SPs, where n is the number of SPs in the system.

4.3.2 Skewed Query Model

Keyword popularity is defined as the fraction of queries associated with the keyword. Suppose that the system receives a query consisting of several keywords. The possibility of appearance of a keyword in the query can be estimated by two factors: i.e., 1) the probability of query length, and 2) the probability of associating each keyword to the query.

In the model, the length of each query is assumed to follow a binomial distribution as is roughly verified using monthly statistical data [85]. More concretely, we assume that each query is constructed using a Bernoulli trial with stopping probability p; i.e., the probability that a given query has length i is given by as follows:

$$L_i \overset{\text{def}}{=} (1-p)^{i-1} \times p. \tag{4.1}$$

Let Q denote the set of queries, and L be the maximum query length. Then, according to the basic keyword search model described in Section 4.3.1, the number of sub-queries generated in the system is

$$|Q| \times \sum_{1 \le i \le L} (1-p)^{i-1} \times p \times i. \tag{4.2}$$

Let p_j denote the probability of associating keyword d_j to a query, and we assume that it follows the Zipf's first law, i.e.,

$$p_j = \frac{1}{i^s \times H_{|D|,s}}, \tag{4.3}$$

where

$$H_{|D|,s} \overset{\text{def}}{=} \sum_{1 \le j \le |D|} (1/i^s), \tag{4.4}$$

and s is a parameter called Zipf-parameter. Note that $\sum_{j=1}^{|D|} p_j = 1$ by definition, where D is the super-set of keywords in the system.

It should be worth noting that under such natural and reasonable assumptions, a number of sub-queries will be generated for getting indices matched to the key of sub-queries. The number of sub-queries associated with each keyword will be highly imbalanced; i.e., a popular keyword is contained in a large number of queries, while an unpopular keyword is rarely contained in the queries.

To better understand this problem, we have performed a simulation to gather sub-query information of SPs in the system. At each SP, we keep track of the number of sub-queries received by local index, as well as the number of sub-queries forwarded to other SPs. In the simulation, we fix N=1000, and $|D|$=10000. The total number of queries are 20000 with Zipf's parameter s=0.8. Query length parameter p=0.6 and L=10. Each query is initiated by a randomly selected SP in the system. Figure 4.2 shows the result of sub-queries received and forwarded at each SP. In this figure, it is found that 1) a large number of SPs in the system forwarded much more sub-queries than they received; 2) some SPs received large number of sub-queries, but they forwarded few sub-queries. 3) some SPs received many sub-queries and also forwarded many sub-queries. These observations justify the importance of routing load reduction and query load balancing in realizing low latency in index retrieval.

4.4 Task Replication based Load Balancing Schemes

In this section, we first introduce some notations using in the Chapter. Afterwards, we propose three concrete replication schemes for balancing query load, i.e., 1) an active index replication scheme to highly decrease routing load in the system, and share response load received by popular indices with replica SPs; 2) a proactive pointer replication scheme to reduce the number of index replicas generated by the first scheme; 3) a passive index replication scheme to guarantee the maximum query load of SPs.

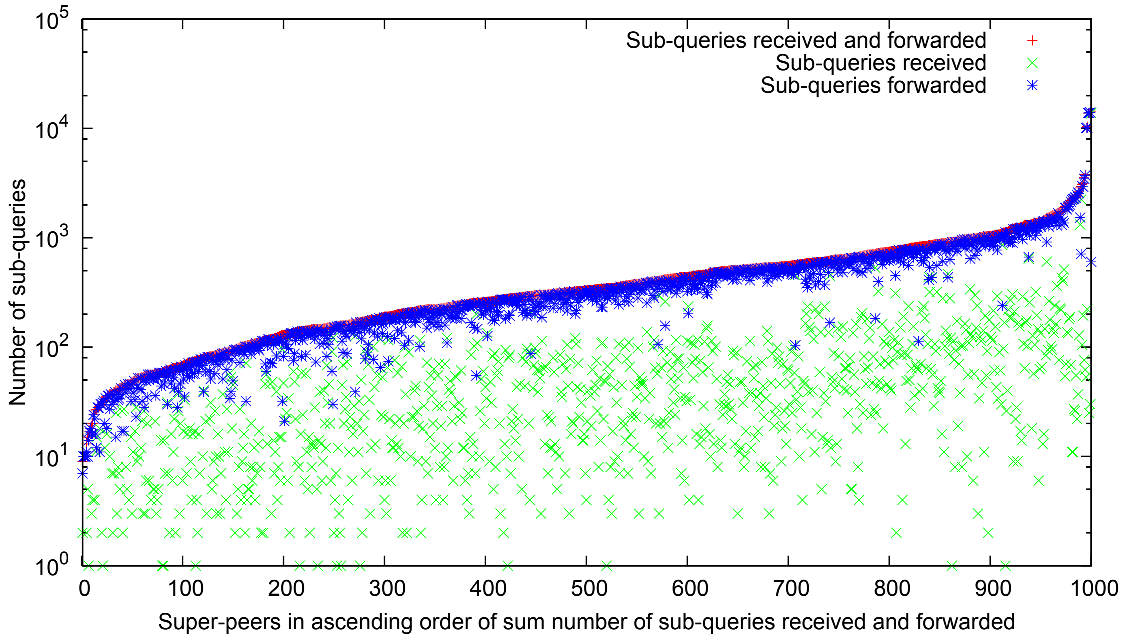


Figure 4.2: Sub-queries received and forwarded at each SP

4.4.1 Preliminaries

Reminding that $D = \{d_1, d_2, \ldots, d_{|D|}\}$ is the superset of keywords in the system. In the following, d_i is also used for representing a sub-query contained keyword d_i (i.e., sub-query d_i) or an index associated with keyword d_i (i.e., index d_i). Let $w(d_i, S_j)$ denote the number of sub-queries received by S_j that stores index d_i (i.e., response load), and let $v(d_i, S_j)$ denote the number of sub-queries initiated or forwarded by S_j for index d_i (i.e., routing load). $w(d_i, S_j)$ and $v(d_i, S_j)$ are periodically calculated by every SP during a unit time interval T. Exponential moving average (EMA) technique is employed to predict the traffic in the next time period, the formula for calculating $w(d_i, S_j)$ and $v(d_i, S_j)$ of S_j at time periods $t \geq 2$ is

$$\begin{aligned} w^t(d_i, S_j) &\overset{\text{def}}{=} \beta w^{t-1}(d_i, S_j) + (1-\beta) w^t(d_i, S_j), \\ v^t(d_i, S_j) &\overset{\text{def}}{=} \beta v^{t-1}(d_i, S_j) + (1-\beta) v^t(d_i, S_j), \end{aligned} \tag{4.5}$$

where $\beta \in [0,1]$. Note that smaller β makes the new observations relatively more important than larger β, while larger β can alleviate replica fluctuation by a suddenly short-time traffic variation. An appropriate value of β can be determined according to the actual situation of the system.

Let $\hat{D}$ be the set of indices and represented as a set of keywords assigned to S_j, and let $\bar{D}$ be the set of sub-queries and represented as a set of keywords initiated or routed by S_j. The load of S_j, denoted as $l(S_j)$, is the weighted sum of its response load and routing load:

$$\ell(S_j) \overset{\text{def}}{=} a \sum_{1 \le i \le |\hat{D}|} w(d_i, S_j) + b \sum_{1 \le i \le |\bar{D}|} v(d_i, S_j). \tag{4.6}$$

In the following, we assume that weighted factor $a = b = 1$, i.e., the entire of routing load is higher than the entire response load in the system. For any given S_i, we also define an threshold value, T_0, which is referred to target load. The target load of SPs which is typically defined as follows:

$$T_0 \overset{\text{def}}{=} \sum_{S_j \in N} \ell(S_j) + c \tag{4.7}$$

where $c > 0$ is an appropriate slack.

4.4.2 Active Index Replication

Under the skewed query model, if we could make a substantial reduction on the search hops for popular indices, routing load in the system would be highly reduced, and the response load of home SPs that stored popular indices will be shared with index replica SPs.

The key idea of the first scheme is to bound the maximum response load of each index by an appropriate constant μ. With this constraint, an index with a popular keyword need to have more replicas to distribute the response load of the index.

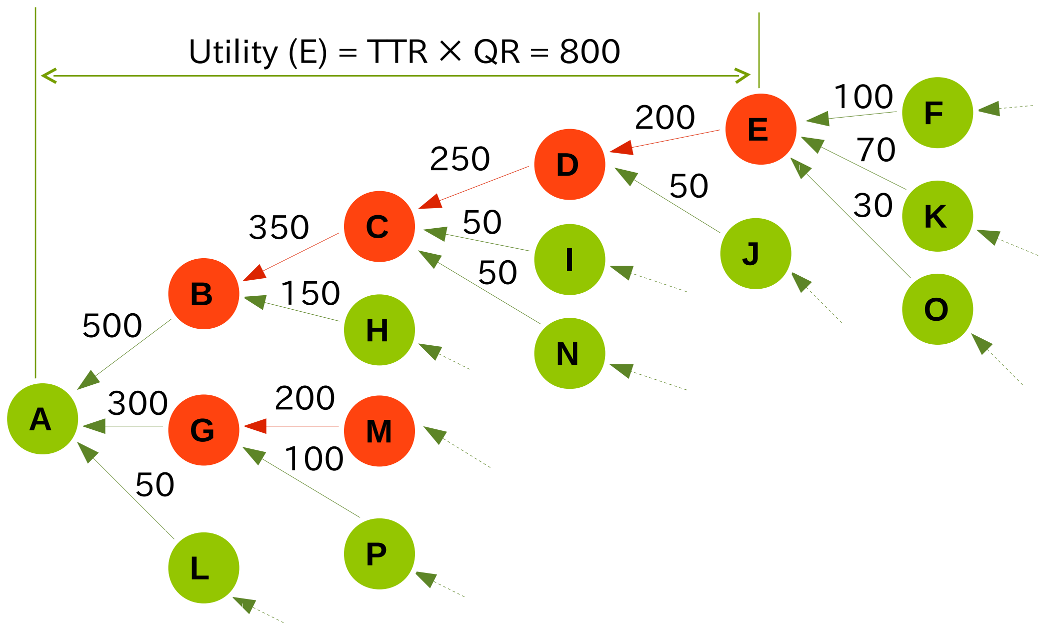


Figure 4.3: Query flow in tree-based routing DHT

Suppose that S_i has an index d_i, a bottom-up directed weighted tree graph rooted at S_i can be utilized for describing query flow of searching d_i. The search from a SP in the tree is the process of moving sub-query d_i along the path towards to root. The level of a SP in the tree is the path length from such SP to root, where the level of root is 0. In the tree graph, each SP maintains the number of incoming queries for index d_i from its child SPs.

Load information aggregation is conducted using this tree. In order to ensure utilization of replicas, we exclude some SPs with low traffic rate to be candidate SPs of index replicas by setting an appropriate minimum load threshold $T_{min} = \alpha\mu$, where $\alpha \in [0, 1]$. Each SP S_x in the tree forwards load information of a child SP $(v(d_i, S_j), TTR)$ to its parent SP, if and only if the child S_j satisfies two condition: i) $v(d_i, S_j) \geq T_{min}$; and ii) $v(d_i, S_j)$ is the biggest among S_x's child SPs. TTR is initialized to 1 and used for recording the level of S_j. In such a way, home SP S_i will maintain R disjoint paths, which is referred to as *critical path* of home SP. Let N_i be the direct child SP set of home SP S_i, then the number of critical paths will be no more than the size of home SP's child SPs, i.e., $|R| \leq |N_i|$. An example of query flow for an index is given in Figure 4.3. In the Figure 4.3, when $T_{min} = 200$, A maintains two critical paths, i.e., (B, C, D, E) and (G, M).

The concrete process of index replication initiated at home SP or replica SP that stored d_i is given in Algorithm 1. As shown in Algorithm 1, index replication is triggered when response load received by d_i at S_j exceeds a threshold μ (line 1), this operation is periodically executed by S_j. Each time the threshold is reached, a critical path will be selected if the direct child SP of home SP S_i has the highest routing load for index d_i (lines 3-5). Upon determining the critical path, a utility value of placing replica on each SP in the critical path is calculated, where the utility value means the reduction of routing load if placing a replica on that SP. The SP corresponding to the highest utility value will be selected to receive a replica of index d_i (line 6-7). The above process will be repeatedly executed until response load $w(d_i, S_i)$ is no more than μ (2-8).

Replica deletion is a reverse operation of replica placement, which is effective to reduce maintenance cost of replicas. We propose a lightweight replica deletion algorithm executed by each replica SP. In replica placement, each replica is attached a TTL field, where TTL is the retention period of the index replica. Remainder that T is the unit time interval of calculating the traffic rate. Thus, TTL can be divided into a number of unit time intervals by T. Let's denote $w(d_i, S_j) < \delta T_{min} (\delta < 1)$, where δ is a underloaded factor, indicates the replica as infrequently used replica. By checking the last h unit time intervals of TTL, if $w(d_i, S_j) < \delta T_{min}$ condition continuously occurs during h time intervals, the index replica will be automatically deleted after TTL is exhausted. Otherwise, TTL will be automatically reset to the initial value and consequently the index replica will be kept for a new retention period.

Algorithm 1 Pseudocode of active index replication of d_i, executed at home SP or replica S_i

```
Ensure: $w(d_i, S_i) \leq \mu$
1: if $w(d_i, S_i) > \mu$ then
2:     repeat
3:         for all path $r_i \in R$ do
4:             Choose the path $r_i$ satisfying that $r_i \leftarrow \arg\max_{n_k \in N_i}(n_k, r_i)$.
5:         end for
6:         $d_i, S_j \leftarrow \arg\max_{S_j \in r_i} TTR \times v(d_i, S_j)$.
7:         send a replica of indices associated with $d_i$ to $S_j$.
8:     until $w(d_i, S_i) \leq \mu$
9: end if
```

Above all, the determination of keeping index replicas can be executed by each replica SP without extra communication cost.

4.4.3 Proactive Pointer Replication

The second scheme places a number of pointers for each key (i.e., hashed keyword), in order to reduce the number of index replicas generated in the first scheme. Note that the pointer is a pair of a keyword and its home SP.

Intuitively, each keyword d_i is assigned to a predetermined number of identified points in a ID space. The corresponding SPs of these identified points are referred to as "gateway" SPs. The gateway SPs divide the ID space into approximately equivalence sub-ranges, and each of which maintains a pointer to the home SP of d_i. By having all queries initiated by SPs from a sub-range passing through the gateway SP in the sub-range, the height of the query flow tree for searching index d_i can be compressed.

A gateway SP can monitor the traffic coming from its sub-range, and then help home SP placing more replicas to the high traffic of sub-ranges, thereby effectively reduce the routing load in the system. Moreover, increased disjoint paths towards home SPs could further balance query load of SPs.

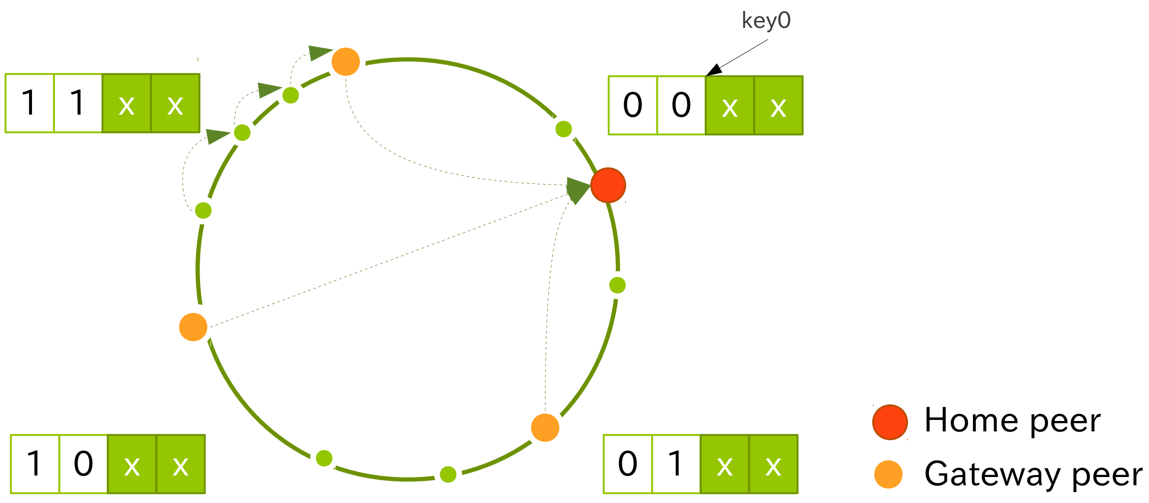


Figure 4.4: Query routing after applying pointer replication

More concretely, the idea of the second scheme is realized by introducing a system parameter b. By changing the b-bits prefix of the hashed d_i, $2^b - 1$ identified points are determined. The home SP of d_i places pointers on the corresponding SPs (i.e., gateway SP) of these identified points. Then in the query routing, each requester selects the closest gateway SP based on the ID space distance between the search key and the requester SP. An example of query routing using pointers based on Chord is shown in Figure 4.4. As shown in Figure 4.4, the hash value of “key0” is 00xx. In the case of $b = 2$, three gateway SPs corresponding to 01xx, 10xx, 11xx are fixed. An requester in the range between 10xx and 11xx, could calculate the distance between itself and 00xx, 01xx, 10xx and 11xx, respectively. Then, it will choose the closest gateway SP in the clockwise direction, i.e., the SP corresponding to 11xx. Finally, the gateway SP will forward the query to home SP using the information in pointer replica.

The concrete process of placing pointer replicas based on Chord is given in Algorithm 2. First, let's prepare a null set K and let m-bit k^* be the hash value of keyword d_i (line 1). Then, S_i changes the b-bits prefix of k^* using bit operation function ShiftRight and ShiftLeft and then move 2^b keys into K (line 2-6). Last, S_i will send a pointer replica to every corresponding SP who is the successor of $k \in K$ except S_i itself (line 7-9).

Algorithm 2 Pseudocode of proactive pointer replication for d_i, executed by home SP S_i

```
1: Let $K = \emptyset$, and $k^* = hash(d_i)$.
2: Preserve low $(m - b)$ bits of $k$, saved as $l$.
3: ShiftRight$(m - b, k)$, saved as $h$.
4: for all $i \in [0, 2^b]$ do
5:     ShiftLeft$(m - b, h \oplus i) + l$, and move it to $K$.
6: end for
7: for all $k \in K \setminus k^*$ do
8:     Home SP $S_i$ send a pointer replica to the SP corresponding to $k$.
9: end for
```

4.4.4 Passive Index Replication

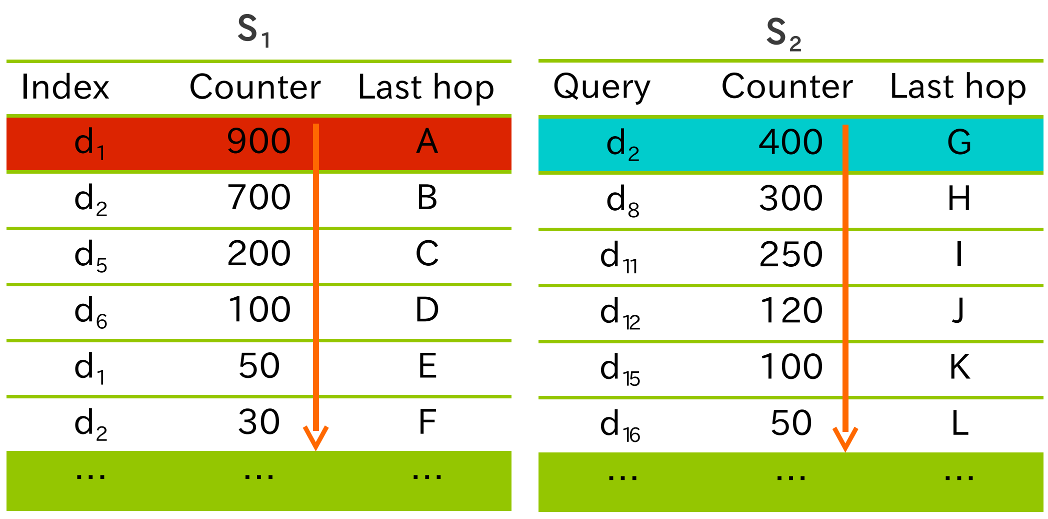
In the system, a SP would be still overloaded caused by the summation of response load and routing load , although the imbalance of query load of SPs is alleviated after applying Scheme 1 and Scheme 2. In order to solve this problem, we distinguish overloading of SPs caused mainly by the routing load and the respond load, and effectively determine the operation of pushing or pulling index replicas under each case.

Algorithm 3 Pseudocode of excessive load shedding, after the query load of S_i exceeds T_0

```
Ensure: $\ell(S_i) \leq T_0$
 1: $query\_counter \leftarrow query\_counter + 1$
 2: if $query\_counter > T_0$ then
 3:     repeat
 4:         if $\sum_{1 \leq i \leq |\hat{D}|} w(d_i, S_j) \geq \sum_{1 \leq i \leq |\bar{D}|} v(d_i, S_j)$ then
 5:             $d_i, S_j \leftarrow \arg\max_{d_i \in \hat{D}, S_j \in N_i} w(d_i, S_j)$
 6:             push a replica of indices associated with $d_i$ to neighbor $S_j$
 7:         else
 8:             $d_i, S_j \leftarrow \arg\max_{d_i \in \bar{D}, S_j \in N_i} v(d_i, S_j)$
 9:             pull a replica of indices associated with $d_i$ from home SP
10:             push the replica of indices associated with $d_i$ to neighbor $S_j$
11:         end if
12:     until $\ell(S_i) \leq T_0$
13: end if
```

The concrete process of load shedding is given in Algorithm 3. A query counter for counting total number of incoming queries is used at every SP (line 1). If S_i exceeds the threshold of query load T_0 (line 2), then the following process is to justify the main cause of this overloading (line 4-11). If this overloading is caused mainly by response load (line 4), S_i will choose an index replica of d_i to its neighbor S_j, where d_i and S_j are selected by sorting $w(d_i, S_j)$ maintained by S_i (line 5-6). For example, in Figure 4.5, S_1 will push a replica of index d_1 to neighbor A. Otherwise, S_i will pull an index replica from home SP that stored index d_i, and then push the index replica to its neighbor S_j, where d_i and S_j are selected by sorting $v(d_i, S_j)$ maintained by S_i

(line 8-10) For example, in Figure 4.5, S_2 will pull a replica of d_2 from home peer S_1, and then push it to neighbor G. The above process will be repeatedly executed until query load $l(S_i)$ is no more than T_0 (3-12).



S_1

Index	Counter	Last hop
d_1	900	A
d_2	700	B
d_5	200	C
d_6	100	D
d_1	50	E
d_2	30	F
...	...	...

S_2

Query	Counter	Last hop
d_2	400	G
d_8	300	H
d_{11}	250	I
d_{12}	120	J
d_{15}	100	K
d_{16}	50	L
...	...	...

Figure 4.5: Load shedding distinguished by main cause of overloading

The purpose of our third scheme is to eliminate bottleneck situation of query processing in the system. Thus, an appropriate target load T_0 should be determined according to the actual situation of the system. It is worth noting that there is a trade-off between the target load and index replication cost incurred by the third scheme. If the target load is set with a high slack value c, the maximum query load of SPs would be still high, and consequently such SPs would become bottleneck of query processing in the system. On the other hand, if the target load is set with a small slack value c, a large number of index replicas would be created, thereby maintenance cost of these index replicas would be high in the system.

4.5 Simulation

We evaluated our proposed schemes based on Overlay Weaver [77]. Chord is used as the routing protocol in our simulation. Chord assigns each SP a 160-bit identifier using a SHA-1 hash function. Each SP maintains its successor, predecessor, and finger table with 160 entries. Note that the size of finger table would be incomplete when the system stabilization process of Chord is not conducted enough times. In our simulations, the stabilization period is set from 2s to 128s. After all SPs joined the system, the system wait 1h for letting SPs fix their finger tables before index uploading and query searching.

The following simulations are conducted to verify the effect of index replication. Thereby, we assume that each of index size is equivalent (e.g. 200KB), and we measure the cost of index replication in terms of the number of index replicas in the system. In other words, we do not simulate the actual maintenance cost of index replicas, and we do not simulate the size of each object. Finally, the cost of pointer replication is ignored in the simulation, since the size of pointer replica is small enough.

4.5.1 Simulation Model

In the following, we fix N=1000, and $|D|$=50, i.e., we do not consider join and leave of SPs. The probability of associating keyword set from D to a query follows the Zipf's distribution with parameter $s = 0.8$. The total number of queries are fixed to 20,000. The length of each query is determined by a Bernoulli trial with stopping probability 0.4, i.e., the probability of assigning i tags is given by $(0.6)^{i-1} \times 0.4$, while the maximum length of each query is bounded by 10. The requester are chosen follows a bounded Pareto distribution with parameter 2. The time interval of generating queries is $20ms$. The unit time interval $T = 1s$, and the EMA parameter $\beta = 0.5$. The system parameter $b = 4$. Note that the weight of response load and routing load are fix to 1. Finally, we do not set any restriction on storage capacity of SP, i.e., there is no replica will be discarded due to the storage usage in the following simulations.

4.5.2 Effect of Replication Schemes

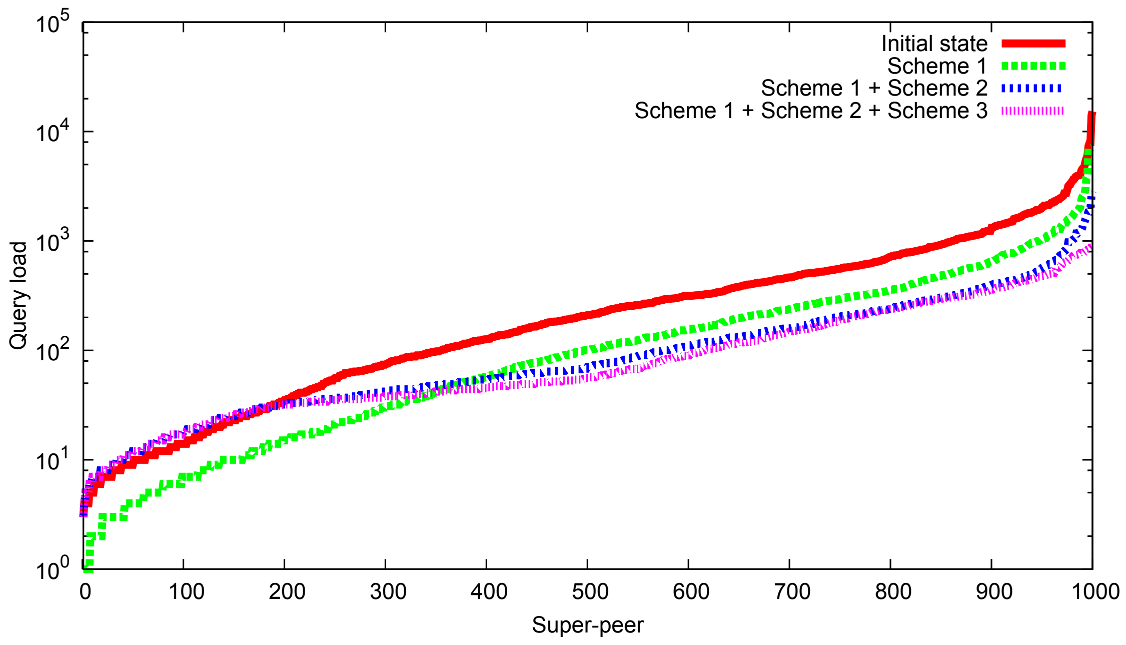


Figure 4.6: Load distribution after applying replication schemes

We first evaluate the performance of three replication based query load balancing schemes. Figure 4.6 summaries the result. The vertical axis is the query load of each SP and the horizontal axis is SPs arranged in an ascending order of the query load. Four curves in the figure show the initial query load distribution and the query load distributions after applying three schemes, respectively. Empirically, we set $\mu = 500$ and $T_{min} = 80$ in Scheme 1. The reason of determining $T_{min} = 80$ can be explained using Table 4.2. In Table 4.2, we investigate the effect of T_{min} by varing T_{min} form 30 to 180 with an interval of 50. We found that when T_{min} is smaller than 80, it cannot significantly reduce the replication cost anymore. While a smaller T_{min} means more load aggregation messages needed in the system. Thus, we set $T_{min} = 80$ in Scheme 1. In addition, the target query load $T_0 = 1000$.

Table 4.2: Effect of parameter T_{min}

T_{min}	Number of Index Replicas
180	175
130	158
80	135
30	134

As shown in Figure 4.6, by applying Scheme 1, the query load (more exactly, routing load) of system is significantly reduced by 47.26% compared to initial state. Meanwhile, the load variation is significantly reduced from 1055.79 to 627.52. Further, by combining Scheme 1 and Scheme 2, the query load is reduced by 68.28% compared to initial state, and the load variation is reduced to 262.44. In the Figure, it is easy to see that the query load of SPs is more balanced after applying Scheme 2. Finally, by applying Scheme 3, the query load is reduced by 73.95% compared to initial state, and the load variation is further reduced to 165.69.

Table 4.3 shows the cost of replication based load balancing schemes in terms of the number of index replicas. This result indicates that by applying Scheme 2, it can reduce the number of index replicas generated by Scheme 1. Moreover, by applying Scheme 3, query load of all SPs are smaller than target load.

Consequently, we could attain a sufficient query load reduction by Scheme 1. After applying Scheme 2, we could get more even load distribution compared to Scheme 1 with a smaller cost. There are few overloaded SPs which could not be solved by Scheme 1 and Scheme 2, but could be effectively resolved by Scheme 3 with acceptable

Table 4.3: Cost of replication

Scheme	Number of Index Replicas
Initial State	NULL
Scheme 1	135
Scheme 1 and 2	98
Scheme 1, 2 and 3	120

cost.

4.5.3 Comparison with PCache Replication

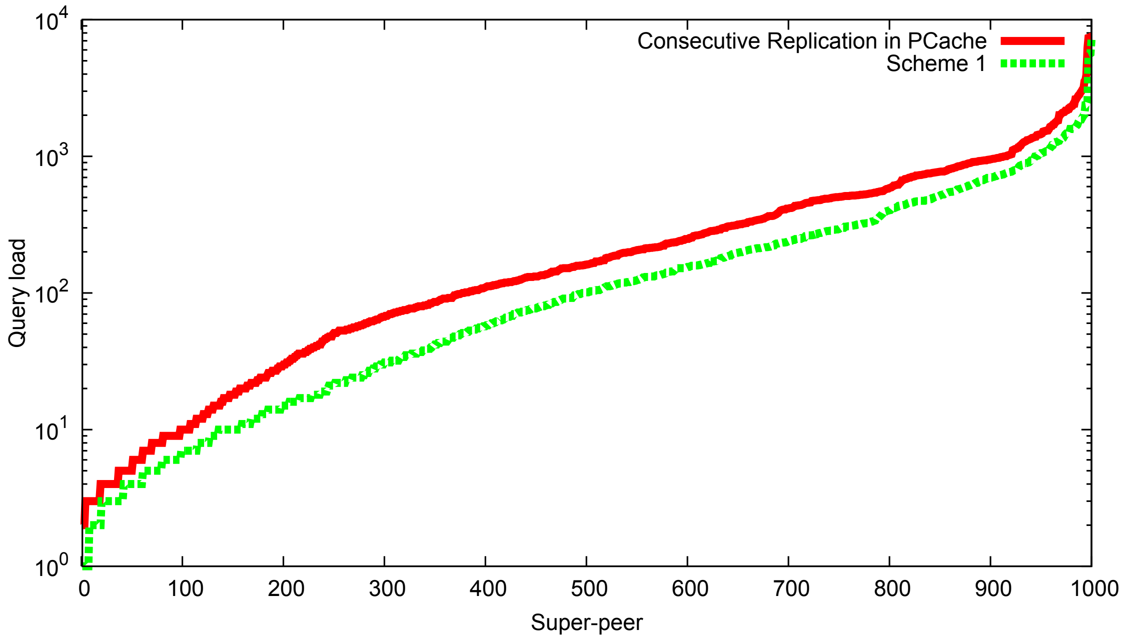


Figure 4.7: Comparison with PCache

In this section, we compare our replica placement schemes with PCache replication. To be fairly comparable, we evaluate the effect of query load distribution when the given number of replicas are equal to 80. Figure 4.7 illustrates the result.

As shown in Figure 4.7, Scheme 1 can further reduce 34.36% query load compared to PCache. The load variation of PCache is 713.69, higher than 513.25 in Scheme 1. It is because in each index replication operation, PCache place the index replicas consecutively at the predecessor SPs without considering of the actual query rate of replica SPs.

4.5.4 Comparison with EAD Replication

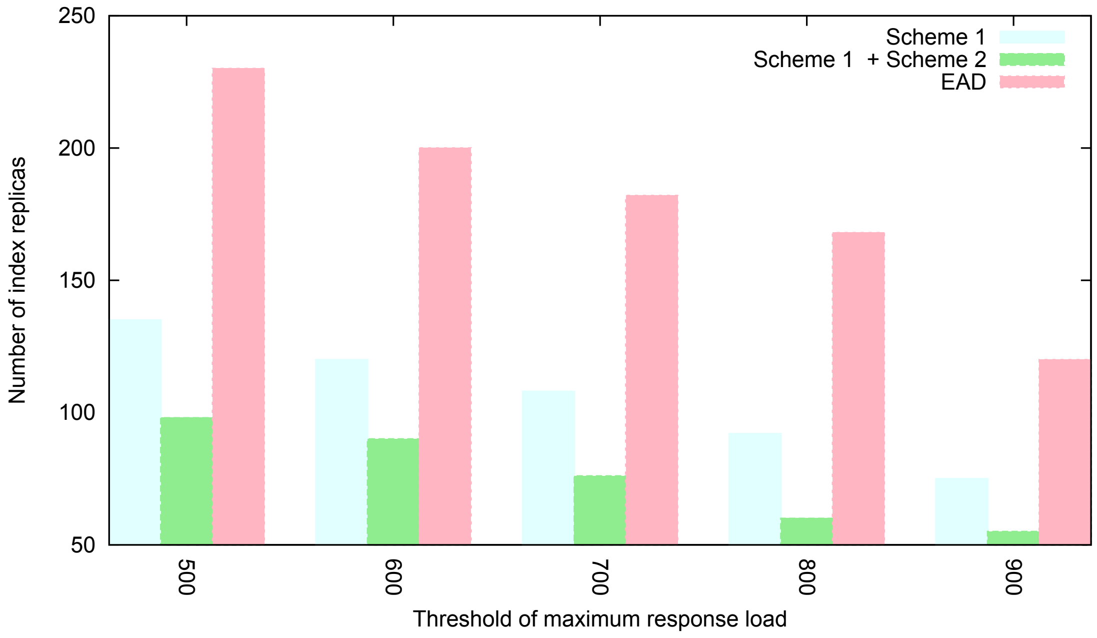


Figure 4.8: Comparison with EAD

In this section, we compare our replica placement schemes with EAD replication. Under a given target load T_0, we evaluate the index replication cost in terms of the number of index replicas. Figure 4.8 illustrates the index replication cost of different schemes. By varying the threshold of response load μ from 500 to 900, we found that the number of index replicas generated by EAD is higher than our schemes. It is worth noting that when μ is set smaller, the cost of EAD become significantly higher than our schemes. It is because that EAD selects candidate replica SPs only according to query rate, resulting in a large number of replicas placed near the home SP. In contrast, in Scheme 1, we used TTR to record the level of the candidate replica SPs, and thereby we can select candidate replica SP more effectively on reducing routing load than EAD.

4.6 Summary and Discussion

In this Chapter, we studied the query load balancing problem in DHT-based super-peer architecture, and proposed an integrated solution that consists of three replication schemes to solve it. We evaluated the performance of the proposed schemes by simulation, the results demonstrate that all the schemes can work in coordinate, in avoiding overloading super-peers and alleviating the query load imbalance for the systems. Moreover, by comparison with two prominent replica placement schemes, it was proved that the cost-effectiveness of the proposed schemes.

We have some practical consideration on replication schemes. Firstly, an index replica with elastic size should be implemented. In a practical system, an index for a keyword should be divided into several pages, and those pages are ranked according to an appropriate ranking scheme, as in web systems. When replicating an index, there is no necessary to replica the whole pages of the index. For example, just replicate the first 10 pages, while maintaining a pointer to the home SP of the index for the left pages, since majority of users often browse only first few pages in index retrieval. Secondly, the maintenance cost of index replicas should be considered in practical systems. Thirdly, the performance of replication schemes needs to be measured in a more realistic P2P environment such as heterogeneous bandwidth of super-peers, and join and leave of super-peers.

Chapter 5

Concluding Remarks

5.1 Summary

In this thesis, we proposed novel solutions for managing the load for hierarchical P2P file search, in order to decrease average and variation of query response time during index retrieval process.

In Chapter 1, we introduced the load problem in P2P systems. Meanwhile, existing P2P architectures and a number of well-known search algorithms are introduced.

In Chapter 2, related works to realize load balancing in P2P systems are presented. We classified load balancing solutions into two categories, i.e., task migration and task replication. For each of category, we presented the most relevant load balancing schemes, including their attributes, advantages and drawbacks.

In Chapter 3, we described details of Qin's hierarchical P2P architecture. We have proposed two efficient task migration schemes for this architecture, in alleviating the bottleneck situation of super-peers. The first scheme controls the load of each task in order to decrease the total cost of task migration. The second scheme directly balances the load over tasks by reordering the priority of tags used in the query forwarding step. The effectiveness of the proposed schemes are evaluated by simulation.

In Chapter 4, we described details of DHT based super-peer architecture. We have proposed an integrated solution consisting of three replication schemes for this architecture, in alleviating query load imbalance of super-peers while minimizing the cost. The first scheme is an active index replication in order to decrease routing load in the super-peer layer, and distribute response load of an index among super-peers that stored the replica. The second scheme is a proactive pointer replication that places location information of an index, for reducing maintenance cost between the index and its replicas. The third scheme is a passive index replication that guarantees the maximum query load of super-peers. The result of simulations indicates that the proposed solution is more cost-effective on placing replicas than other schemes.

5.2 Contributions

Our load management solutions focused on two kind of super-peer based hierarchical architectures of P2P systems, which are distinguished by the organization of super-peers. The super-peers can be isolated or organized into a overlay, depending on the design requirement of real systems. For each of architecture, our load balancing solutions provide very good performance.

We summarize the contributions of this thesis as follows:

- We gave a deep observation on two kinds of hierarchical P2P architectures, i.e., Qin's hierarchical P2P architecture and DHT-based super-peer architecture. We analyzed load problem and designed load metric for each kind of architectures.
- we proposed efficient and effective solutions to overcome load problems in these architectures. The proposed solutions are based on task migration and task replication. We evaluated the proposed solutions in simulation. The simulation results proved that our proposed solutions can alleviate load imbalance of super-

peers.

5.3 Discussion and Outlook

There are some possible file sharing applications can be built on Qin's hierarchical P2P architecture and DHT-based super-peer architecture. For example,

- Company wide P2P systems. Super-peer in each geographically distributed subsidiary are controlled by a central server at headquarter.
- University wide P2P systems. Each department has its own super-peer, and connected by DHTs.
- Commercial search engines. Commercial P2P search engines can improve system performance using load balanced super-peer based hierarchical architectures.

Our future work is as follows. At first, our skewed query model assumed in Chapter 3 and Chapter 4 are entirely artificial. We would like to measure the system performance of real queries generated by real users. Secondly, we should improve availability of the proposed solutions in practical systems. For example, a practical system may adopt paging and ranking algorithm of indices into super-peers. Thus, load balancing solutions need to adapt to changes of the system. At last, we should enhance the dependability of the proposed solutions in a more realistic dynamic P2P environment such as heterogeneous bandwidth of super-peers, and join and leave of super-peers.

Bibliography

[1] L. A. Adamic, R. M. Lukose, A. R. Puniyani and B. A. Huberman, "Search in Power-Law Networks," *Physical Review E*, vol. 64, no. 4, 2001, pp.46135-46143.

[2] E. Adar and B. A. Huberman, "Free Riding on Gnutella," *First Monday*, vol.5, no.10, 2000.

[3] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, vol. 36, no. 4, December 2004, pp.335-371.

[4] J. Aspnes and G. Shah, "Skip Graphs," *ACM Transactions on Algorithms*, vol. 3, no. 4, 2007, article no. 37.

[5] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras and F. Silvestri, "Challenges on Distributed Web Retrieval," *the 23rd International conference on Data Engineering*, Istanbul, Turkey, 2007, pp.6-20.

[6] G. Barish and K. Obraczke, "World Wide Web Caching: Trends and Techniques," *IEEE Communications Magazine*, vol.38, no.5, 2000, pp.178-184 .

[7] A. R. Bharambe, M. Agrawal and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," *Proceedings of the 2004 Conference on*

Applications, Technologies, Architectures, and Protocols for Computer Communications, Oregon, USA, 2004, pp.353-366.

[8] S. Bianchi, S. Serbu, P. Felber, and P. Kropf, "Adaptive Load Balancing for DHT Lookups," *Proceedings of 15th International Conference on Computer Communications and Networks*, Arlington, VA, October, 2006, pp.411-418.

[9] N. Bisnik and A. Abouzeid, "Modeling and Analysis of Random Walk Search Algorithms in P2P Networks," *Second International Workshop on Hot Topics in Peer-to-Peer Systems*, 2005, pp.95-103 .

[10] BitComet, `http://www.bitcomet.com/`

[11] BitTorrent, `http://www.bittorrent.com/`

[12] J. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for DHTs," *2nd International Workshop on Peer-to-Peer Systems*, 2002, pp.80-87.

[13] V. Cardellini, M. Colajanni and P. S. Yu, "Dynamic Load Balancing on Web-Server Systems," *IEEE Internet Computing*, vol. 3, no. 3, May-June 1999, pp.28-39.

[14] V. Cardellini, E. Casalicchio, M. Colajanni and P. S. Yu, "The State of the Art in Locally Distributed Web-Server Systems," *ACM Computing Surveys*, vol. 34, no. 2, June 2002, pp.263-311.

[15] V. Cardellini, M. Colajanni and P. S. Yu, "Request Redirection Algorithms for Distributed Web Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, April 2003, pp.355-368.

[16] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham and S. Shenker, "Making Gnutella-like P2P Systems Scalable," *Proceedings of the 2003 Conference on*

Applications, Technologies, Architectures, and Protocols for Computer Communications, Karlsruhe, Germany, 2003, pp.407-418.

[17] I. Clarke, O. Sandberg, B. Wiley and T.W. Hong, “Freenet: A Distributed Anonymous Information Storage and Retrieval System,” *Proceedings of International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*, July 2000, pp.46-66.

[18] E. Cohen and S. Shenker, “Replication Strategies in Unstructured Peer-to-Peer Networks,” *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, October 2002, pp.177-190.

[19] G. Cybenko, “Dynamic Load balancing for Distributed Memory Multiprocessors,” *Journal of Parallel and Distributed Computing*, vol. 7, no. 2, October 1989, pp.279-301.

[20] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area Cooperative Storage with CFS,” *Proceedings of 18th ACM Symposium on Operating System Principles*, New York, NY, 2001, pp.202-215.

[21] N. Daswani, H. Garcia-Molina and B. Yang, “Open Problems in Data-Sharing Peer-to-Peer Systems,” *International Journal of Database Management Systems*, 2003, pp.1-15.

[22] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman and B. Weihl, “Workload-Aware Load Balancing for Clustered Web Servers,” *IEEE Internet Computing*, vol. 6, no. 5, September 2002, pp.1089-7801.

[23] Faroo, `http://www.faroo.com/`

[24] P. Ganesan, M. Bawa and H. Garcia-molina, "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems," *Proceedings of the 30th international conference on Very large data bases*, Toronto, Spain, 2004, pp.444-455.

[25] L. Garces-Erice, E. W. Biersack, P. A. Felber, K. W. Ross, and G. Urvoy-Keller, "Hierarchical Peer-to-Peer Systems," *Proceedings of International Conference on Parallel and Distributed Computing*, 2003.

[26] P. Garbacki, D. H. J. Epema and M. van Steen, "Optimizing Peer Relationships in a Super-Peer Network," *Proceedings of 27th International Conference on Distributed Computing Systems*, Toronto, Canada, 2007, pp.31-42.

[27] M. R. Garey and D. S. Johnson, "Computers and Intractability : A Guide to the Theory of NP-Completeness," *W.H. Freeman and Company*, New York, USA, 1979.

[28] A. Ghodsi, L. O. Alima, and S. Haridi, "Symmetric Replication for Structured Peer-to-Peer Systems," *Proceedings of the 2005/2006 international conference on Databases, information systems, and peer-to-peer computing*, 2005, pp.74-85.

[29] C. Gkantsidis, M. Mihail and A. Saberi "Random Walks in Peer-to-Peer Networks," *Proceedings of 23th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2004.

[30] C. Gkantsidis, M. Mihail and A. Saberi, "Hybrid Search Schemes for Unstructured Peer-to-Peer Networks," *Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, March 2005, pp.1526-1537.

[31] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," *Proceedings of Twenty-third*

AnnualJoint Conference of the IEEE Computer and Communications Societies, 2004, pp.2253-2262.

[32] G. Gonnet, "Expected Length of the Longest Probe Sequence in Hash Code Searching," *Journal of the ACM*, vol. 28, no. 2, April 1981, pp.289-304.

[33] Google, `http://www.google.com`

[34] A. Greenberg, J. Hamilton, D. A. Maltz and P. Patel, "The Cost of a Cloud: Research Problems in Data Center Networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, January 2009, pp.68-73.

[35] C. Harvesf and D. M. Blough, "Replica Placement for Route Diversity in Tree-Based Routing Distributed Hash Tables," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 3, 2011, pp.419-433.

[36] H. V. Jagadish, B. C. Ooi and Q. H. Vu, "Baton: A Balanced Tree Structure for Peer-to-Peer Networks," *Proceedings of 31st International Conference on Very Large Databases Conference*, Trondheim, Norway, 2005, pp.661-672.

[37] I. Jawhar and J. Wu, "A Two-Level Random Walk Search Protocol for Peer-to-Peer Networks," *Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, 2004.

[38] V. Kalogeraki, D. Gunopulos and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," *Proceedings of the Eleventh International Conference on Information and knowledge Management*, 2002, pp.300-307.

[39] D. R. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, New York, NY, 2004, pp.36-43.

[40] Y. Kawasaki, N. Matsumoto and N. Yoshida, "Popularity-Based Content Replication in Peer-to-Peer Networks," *Proceedings of the 6th international conference on Computational Science*, 2006, pp.436-443.

[41] R. A. King, A. Hameurlain, F. Morvan, "Query Routing and Processing in Peer-to-Peer Data Sharing Systems," *Lecture Notes in Computer Science*, vol. 2429, vol.2, no. 2, 2010, pp.116-139.

[42] A. Klemm, C. Lindemann and O. P. Waldhorst, "A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks," *IEEE 58th Vehicular Technology Conference, 2003*, 2003, pp.2758-2763.

[43] X. Li and J. Wu, "Searching Techniques in Peer-to-Peer Networks," *Handbook of Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, New York, USA, 2006.

[44] J. Li, B. Cheung and S. Vuong, "A Scheme for Balancing Heterogeneous Request Load in DHT-based P2P Systems," *Proceedings of the Fourth International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, 2007, Article No. 4.

[45] J. Liang, R. Kumar, and K. Ross, "The Kazaa Overlay: A Measurement Study," *Proceedings of 19th IEEE Annual Computer Communications Workshop*, 2004.

[46] W.K. Lin, C. Ye and D. M. Chiu, "Decentralized Replication Algorithms for Improving File Availability in P2P Networks," *Proceedings of Fifteenth IEEE International Workshop on Quality of Service*, 2007, pp.29-37.

[47] Z.H. Liu, M.H. Lin, A. Wierman, S. H. Low and L. L.H. Andrew, "Greening Geographical Load Balancing," *Proceedings of the ACM SIGMETRICS joint*

international conference on Measurement and modeling of computer systems, New York, USA, 2011, pp.233-244.

[48] Q. Lv, P. Cao, E. Cohen, K. Li and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proceedings of the 16th international conference on Supercomputing*, 2002, pp.84-95.

[49] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System based on the XOR Metric," *Lecture Notes in Computer Science*, vol. 2429, 2002, pp.53-65.

[50] S.H. Min, J. Holliday and D.S. Cho, "Optimal Super-peer Selection for Large-scale P2P System," *Proceedings of the 2006 International Conference on Hybrid Information Technology*, Washington, DC, USA, 2006, pp.588-593.

[51] M. D. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," *Ph.D. dissertation*, 1996, University of California at Berkeley.

[52] A. Mondal, K. Goda and M. Kitsuregawa, "Effective Load-Balancing of Peer-to-Peer Systems," *Proceedings of Data Engineering Workshop*, Ishikawa, Japan, March 2003.

[53] A. Montresor, "A Robust Protocol for Building Superpeer Overlay Topologies," *Proceedings of IEEE International Conference on Peer-to-Peer Computing*, CA, USA, 2004, pp.202-209.

[54] Napster, `http://www.napster.com/`

[55] M. Newman, "Power Laws, Pareto Distributions and Zipf's Law," *Contemporary Physics*, vol. 46, 2005, pp.323-351.

[56] D. Novak, "Load Balancing in Peer-to-Peer Data Networks," *Proceedings of 2nd Doctoral Workshop on Mathematical and Engineering Methods in Computer Science*, 2006, pp.151-157.

[57] G. On, J. Schmitt and R. Steinmetz, "The Effectiveness of Realistic Replication Strategies on Quality of Availability for Peer-to-Peer Systems," *Proceedings of the Third International Conference on Peer-to-Peer Computing*, September 2003, pp.57-64.

[58] A. Peytchev, M. P. Couper, S. E. McCabe and S. D. Crawford, "Web Survey Design. Paging Versus Scrolling," *Public Opinion Quarterly*, vol. 70, No. 4, 2006. pp.596-607.

[59] C. G. Plaxton, R. Rajaraman and A. W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment," *Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*, 1997, pp.311-320.

[60] K. P. N. Puttaswamy, A. Sala and B. Y. Zhao, "Searching for Rare Objects using Index Replication," *Proceedings of 27th Conference on Computer Communications - IEEE INFOCOM 2008*, April 2008, pp.1723-1731.

[61] T.T. Qin, Q. Cao, Q.Y. Wei, and S. Fujita, "A Tag-Based Scheme to Realize Real-Time File Search in Hierarchical Peer-to-Peer Systems," *Journal of Information Processing*, vol. 20, no. 2, 2012, pp.463-471.

[62] T.T. Qin, "A Study on Hierarchical Peer-to-Peer Systems to Realize Real-Time File Search," *Ph.D. thesis, Hiroshima University*, 2012.

[63] M. Raab and A. Steger ““Balls into Bins” - A Simple and Tight Analysis” *Lecture Notes in Computer Science*, vol. 1518, 1998, University of California at Berkeley, pp.159-170.

[64] S. Rajasekhar, B. Rong, K. Y. Lai, I Khalil and Z. Tari, “Load Sharing in Peer-to-Peer Networks using Dynamic Replication,” *Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, 2006, pp.1011-1016.

[65] V. Ramasubramanian and E. G. Sirer, “Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays,” *Proceedings of the First Symposium on Networked Systems Design and Implementation*, Berkeley, CA, 2004, pp.8-8.

[66] K. Ranganathan, A. Iamnitchi, and I. Foster, “Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities,” *Proceedings of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2002, pp.376-376.

[67] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, “Load Balancing in Structured P2P Systems,” *Proceedings of Second International Workshop on Peer-to-Peer Systems*, 2003, pp.68-79.

[68] W.X. Rao, L. Chen, A.W.C. Fu, and G.R. Wang, “Optimal Resource Placement in Structured Peer-to-Peer Networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 7, 2010, pp.1045-9219.

[69] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A Scalable Content-Addressable Network,” *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, 2001, pp.161-172.

[70] S. Rieche, L. Petrak, and K. Wehrle, "A Thermal-Dissipation-Based Approach for Balancing Data Load in Distributed Hash Tables," *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, Washington, DC, 2004, pp.15-23.

[71] L. Rong, "Multimedia Resource Replication Strategy for a Pervasive Peer-to-Peer Environment," *Journal of Computers*, vol. 3, No. 4, April 2008, pp.9-15.

[72] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proceedings of 18th Symposium on Operating Systems Principles*, 2001, pp.188-201.

[73] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, London, 2001, pp.329-350.

[74] S. Serbu, "Load Management in Peer-to-Peer Systems: Structures and Algorithms," *PhD Thesis, University of Neuchatel*, April 2010.

[75] H.Y. Shen, "An Efficient and Adaptive Decentralized File Replication Algorithm in P2P File Sharing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 6, 2010, pp.827-840.

[76] H.Y. Shen and C.Z. Xu, "Elastic Routing Table with Provable Performance for Congestion Control in DHT Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 2, 2010, pp.242-256.

[77] K. Shudo, Y. Tanaka, and S. Sekiguchi, "Overlay Weaver: An Overlay Construction Toolkit," *Computer Communications In Special Issue: Foundation of Peer-to-Peer Computing*, vol. 31, no. 2, 2008, pp.402-412.

[78] G. Skobeltsyn, "Query-Driven Indexing in Large-Scale Distributed Systems," *Ph.D. dissertation, EPFL*, 2009.

[79] Skype, `http://www.skype.com/`

[80] SopCast, `http://www.sopcast.com/`

[81] D. Tsoumakos and N. Roussopoulos, "Analysis and Comparison of P2P Search Methods," *Proceedings of the 1st international conference on Scalable information systems*, Article No.25, 2006.

[82] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, 2001, pp.149-160.

[83] S. Tewari and L. Kleinrock, "Proportional Replication in Peer-to-Peer Networks," *Proceedings of 25th International Conference on Computer Communications and Networks*, April 2006, pp.1-12.

[84] S. M. Thampi and C. S. K, "Survey of Search and Replication Schemes in Unstructured P2P Networks," *Network Protocols and Algorithms*, vol. 2, no. 1, 2010, pp.93-131.

[85] Trellian, `http://www.keyworddiscovery.com/keyword-stats.html`.

[86] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks," *Proceedings of the 3rd International Conference on P2P Computing*, 2003, pp.102-109.

[87] V. Viswanathan, "Load Balancing Web Applications," `http://www.onjava.com/pub/a/onjava/2001/09/26/load.html`

[88] Q. H. Vu, B. C. Ooi, M. Rinard and K. L. Tan, "Histogram-Based Global Load Balancing in Structured Peer-to-Peer Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 4, 2009, pp.595-608.

[89] M.H. Willebeek-LeMair and A.P. Reeves, "Strategies for Dynamic Load Balancing on Highly Parallel Computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 9, September 1993, pp.979-993.

[90] Y.L. Xiong, "A Congestion Control Scheme for Wireless Sensor Networks," *Master Thesis, Texas A&M University*, 2006.

[91] C.Z. Xu and F. C. Lau, "Load Balancing in Parallel Computers: Theory and Practice," *Kluwer Academic Publishers Norwell*, MA, USA, 1997.

[92] YaCy, `http://yacy.net/`

[93] Yahoo, `http://www.yahoo.com`

[94] H. Yamamoto, D. Maruta and Y. Oie, "Replication Methods for Load Balancing on Distributed Storages in P2P Networks," *Proceedings of the The 2005 Symposium on Applications and the Internet*, 2005, pp.264-271.

[95] B. Yang and H. Garcia-Molina, "Comparing Hybrid Peer-to-Peer Systems," *Proceedings of the 27th International Conference on Very Large Databases*, September 2001, pp.561-570.

[96] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," *Proceedings of the 22nd IEEE International Conference on Distributed Computing*, 2002, pp.5-14.

[97] B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-Peer Networks," *Proceedings of International Conference on Distributed Computing Systems*, 2002, pp.5-14.

[98] B. Yang, and H. Garcia-Molina, “Designing A Super-Peer Network,” *Proceedings of 19th International Conference on Data Engineering*, March 2003, pp.49.

[99] X.Y. Yang and G. Veciana, “Service Capacity of Peer-to-Peer Networks,” *Proceedings of Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, March 2004, pp.2242-2252.

[100] Q. Zhang, A. Riska, W. Sun, E. Smirni and G. Ciardo, “Workload-Aware Load Balancing for Clustered Web Servers,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 3, March 2005, pp.219-233.

[101] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing, Technical report, University of California at Berkeley, 2001.

[102] J. Zhou, X. Zhang, L. N. Bhuyan, and B. Liu, “Clustered K-Center: Effective Replica Placement in Peer-to-Peer Systems,” *Proceedings of the Global Communications Conference*, Washington, DC, November, 2007, pp.2008-2013.

[103] Y.W. Zhu , H.H. Wang and Y.M. Hu, “A Super-peer Based Lookup in Structured Peer-to-Peer Systems,” *Proceedings of 16 th International Conference on Parallel and Distributed Computing Systems*, 2003, pp.465-470.

Publication

International Journal

- Q. Cao and S. Fujita, “Load Balancing Schemes for a Hierarchical Peer-to-Peer File Search System,” *International Journal of Grid and Utility Computing (IJGUC)*, vol. 2, no. 2, pp. 164-171, 2011.
- Q. Cao and S. Fujita, “Cost-effective Replication Schemes for Query Load Balancing in DHT based Peer-to-Peer File Search,” *Journal of Information Processing Systems (JIPS)*. (Accepted.)

International Conference

- Q. Cao and S. Fujita, “Load Balancing Schemes for a Hierarchical Peer-to-Peer File Search System,” In *Proceedings the Fifth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2010)*, pp. 63-70, November, 2010.