Load Balancing in Peer-to-Peer Systems

Haiying Shen

Computer Science and Computer Engineering Department

University of Arkansas

Fayetteville, Arkansas, USA

Abstract

Structured peer-to-peer (P2P) overlay networks like Distributed Hash Tables (DHTs) map data items to the network based on a consistent hashing function. Such mapping for data distribution has an inherent load balance problem. Thus, a load balancing mechanism is an indispensable part of a structured P2P overlay network for high performance. The rapid development of P2P systems has posed challenges in load balancing due to their features characterized by large scale, heterogeneity, dynamism, and proximity. An efficient load balancing method should flexible and resilient enough to deal with these characteristics. This chapter will first introduce the P2P systems and the load balancing in P2P systems. It then introduces the current technologies for load balancing in P2P systems, and provides a case study of a dynamism-resilient and proximity-aware load balancing mechanism. Finally, it indicates the future and emerging trends of load balancing, and concludes the chapter.

Keywords: ***Peer-to-Peer, Distributed Hash Table, Distributed Systems, Load Balancing, Heterogeneity, Proximity, Locality, Dynamism, Churn.***

Table of Contents

1. Introduction

Peer-to-peer (P2P) overlay network is a logical network on top of a physical network in which peers are organized without any centralized coordination. Each peer has equivalent responsibilities, and offers both client and server functionalities to the network for resource sharing. Over the past years, the immense popularity of P2P resource sharing services has produced a significant stimulus to content-delivery overlay network research (Xu, 2005). An important class of the overlay networks is structured P2P overlays, i.e. distributed hash tables (DHTs), that map keys to the nodes of a network based on a consistent hashing function (Karger, 1997). Representatives of the DHTs include CAN (Ratnasamy, 2001), Chord (Stoica, 2003), Pastry (Rowstron, 2001), Tapestry (Zhao, 2001), Kademlia (Maymounkov, 2002), and Cycloid (Shen, 2006); see (Shen, 2007) and references therein for the details of the representatives of the DHTs.

In a DHT overlay, each node and key has a unique ID, and each key is mapped to a node according to the DHT definition. The ID space of each DHT is partitioned among the nodes and each node is responsible for those keys the IDs of which are located in its space range. For example, in Chord, a key is stored in a node whose ID is equal to or succeeding the key's ID. However, a downside of consistent hashing is uneven load distribution. In theory, consistent hashing produces a bound of O(log n) imbalance of keys between nodes, where n is the number of nodes in the system (Karger, 1997). Load balancing is an indispensable part of DHTs. The objective of load balancing is to prevent nodes from being overloaded by distributing application load among the nodes in proportion to their capacities.

Although the load balancing problem has been studied extensively in a general context of parallel and distributed systems, the rapid development of P2P systems has posed challenges in load balancing due to their features characterized by large scale, heterogeneity, dynamism/churn, and proximity. An efficient load balancing method should flexible and resilient enough to deal with these characteristics. Network churn represents a situation where a large percentage of nodes and items join, leave and fail continuously and rapidly, leading to unpredicted P2P network size. Effective load balancing algorithms should work for DHTs with and without churn and meanwhile be capable of exploiting the physical proximity of the network nodes to minimize operation cost. By proximity, we mean that the logical proximity abstraction derived from DHTs don't necessarily match the physical proximity information in reality. In the past, numerous load balancing algorithms were proposed with different characteristics (Stoica, 2003; Rao, 2003; Godfrey, 2006; Zhu, 2005; Karger, 2006). This chapter is dedicated to providing the reader with a complete understanding of load balancing in P2P overlays.

The rest of this chapter is organized as follows. In Section 2, we will give an in depth background of load balancing algorithms in P2P overlays. We move on to present the load balancing algorithms discussing their goals, properties, initialization, and classification in Section 3. Also, we will present a case study of a dynamism-resilient and locality-aware load balancing algorithm. In Section 4, we will discuss the future and emerging trends in the domain of load balancing, and present the current open problems in load balancing from the P2P overlay network perspective. Finally, in Section 5 we conclude this chapter.

2. Background

Over the past years, the immerse popularity of the Internet has produced a significant stimulus to P2P file sharing systems. A recent study of large scale characterization of traffic (Saroiu, 2002) shows that more than 75% of Internet traffic is generated by P2P applications.

Load balancing is an inherent problem in DHTs based on consistent hashing functions. Karger *et al.* proved that the consistent hashing function in Chord (Karger, 1997) leads to a bound of O(log n) imbalance of keys between the nodes. Load imbalance adversely affects system performance by overloading some nodes, while prevents a P2P overlay from taking full advantage of all resources. One main goal of P2P overlays is to harness all available resources such as CPU, storage, and bandwidth in the P2P network so that users can efficiently and effectively access files. Therefore, load balancing is crucial to achieving high performance of a P2P overlay. It helps to avoid overloading nodes and make full use of all available resources in the P2P overlay.

Load balancing in DHT networks remains challenging because of their two unique features:

1. Dynamism. A defining characteristic of DHT networks is dynamism/churn. A great number of nodes join, leave and fail continually and rapidly, leading to unpredicted network size. A load balancing solution should be able to deal with the effect of churn. Popularity of the items may also change over time. A load balancing solution that works for static situations does not necessarily guarantee a good performance in dynamic scenarios. Skewed query patterns may also result in considerable number of visits at hot spots, hindering efficient item access.
2. Proximity. A load balancing solution tends to utilize proximity information to reduce the load balancing overhead. However, logical proximity abstraction derived from DHTs doesn't necessarily match the physical proximity information in reality. This mismatch becomes a big obstacle for the deployment and performance optimization of P2P applications.

In addition, DHT networks are often highly heterogeneous. With the increasing emergence of diversified end devices on the Internet equipped with various computing, networking, and storage capabilities, the heterogeneity of participating peers of a practical P2P system is pervasive. This requires a load balancing solution to distribute not only the application load (e.g. file size, access volume), but also the load balancing overhead among the nodes in proportion to their capacities.

Recently, numerous load balancing methods have been proposed. The methods can be classified into three categories: virtual server, load transfer and ID assignment or reassignment. Virtual server methods (Stoica, 2003; Godfrey, 2005) map keys to virtual servers the number of which is much more than real servers. Each real node runs a number of virtual servers, so that each real node is responsible for O(1/n) of the key ID space with high probability. Load transfer methods (Rao, 2003; Karger, 2004; Zhu, 2005) move load from heavily loaded nodes to lightly loaded nodes to achieve load balance. ID assignment or reassignment methods (Bienkowski, 2005; Byers, 2003) assign a key to a lightly loaded node among a number of options, or reassign a key from a heavily loaded node to a lightly loaded node.

3. Load Balancing Methods

3.1 Examples of Load Balancing Methods

In this section we will review various load balancing methods that have been proposed for structured P2P overlays over the last few years. For each method, we will review its goals, algorithms, properties, and pros and cons.

3.1.1 Virtual Server

Basic Virtual Server Method. Consistent hashing leads to a bound of O(log n) imbalance of keys between nodes. Karger et al. (Karger, 1997) pointed out that the O(log n) can be reduced to an arbitrarily small constant by having each node run (log n) virtual nodes, each with its own identifier. If each real node runs *v* virtual nodes, all bounds should be multiplied by *v*. Based on this principle, Stoica *et al.* (2003) proposed an abstraction of virtual servers for load balancing in Chord. With the virtual server method, Chord makes the number of keys per node more uniform by associating keys with virtual nodes, and mapping multiple virtual nodes (with unrelated identifiers) to each real node. This provides a more uniform coverage of the identifier space. For example, if (log n) randomly chosen virtual nodes are allocated to each real node, with high probability each of the n bins will contain O(log n) virtual nodes (Motwani, 1995). The virtual server-based approach for load balancing is simple in concept. There is no need for the change of underlying DHTs. However, the abstraction incurs large space overhead and compromises lookup efficiency. The storage for each real server increases from O(log n) to O($\log^2$ n) and the network traffic increase considerably by a factor of Ω(log n). In addition, node joins and departures generate high overhead for nodes to update their routing tables. This abstraction of virtual server simplifies the treatment of load balancing problem at the cost of higher space overhead and lookup efficiency compromise. Moreover, the original concept of virtual server ignores the node heterogeneity.

Y_0 DHT protocol. Brighten *et al.* (Godfrey, 2005) addressed the problem of virtual server method by arranging a real server for virtual ID space of consecutive virtual IDs. This reduces the load imbalance from O(log n) to a constant factor. The authors developed a DHT protocol based on Chord, called Y_0, that achieves load balancing with minimal overhead under the

assumption that the load is uniformly distributed in the ID space. The authors proved that Y_0 can achieve near-optimal load balancing with low overhead, and it increases the size of the routing tables by at most a constant factor.

Y_0 is based on the concept of virtual servers, but with a twist: instead of picking *k* virtual servers with random IDs, a node clusters those IDs in a random fraction $\Theta(k/n)$ of the ID space. This allows the node to share a single set of overlay links among all *k* virtual servers. As a result, the number of links per physical node is still $\Theta(\log n)$, even with $\Theta(\log n)$ virtual servers per physical node. To deal with node heterogeneity, Y_0 arranges higher-capacity nodes to have a denser set of overlay links, and allows lower-capacity nodes to be less involved in routing. It results in reduced lookup path length compared to the homogeneous case in which all nodes have the same number of overlay links. Y_0 leads to more significant than Chord with the original concept of virtual server, because its placement of virtual servers provides more control over the topology.

Real-world simulation results show that Y_0 reduces the load imbalance of Chord from $O(\log n)$ to a less than 3.6 without increasing the number of links per node. In addition, the average path length is significantly reduced as node capacities become increasingly heterogeneous. For a real-word distribution of node capacities, the path length in Y_0 is asymptotically less than half the path length in the case of a homogeneous system.

Y_0 operates under the uniform load assumption that the load of each node is proportional to the size of the ID space it owns. This is reasonable when all objects generate similar load (e.g., have the same size), the object IDs are randomly chosen (e.g., are computed as a hash of the object's content), and the number of objects is large compared to the number of nodes (e.g., $\Omega(n \log n)$). However, some of the cases may not hold true in reality.

Virtual Node Activation. In virtual server methods, to maintain connectivity of the network, every virtual node needs to periodically check its neighbors to ensure their updated status. More virtual nodes will lead to higher overhead for neighbor maintenance. Karger and Ruhl (Karger, 2004) coped with the virtual server problem by arranging for each real node to activate only one of its O(log n) virtual servers at any given time. The real node occasionally checks its inactive virtual servers and may migrate to one of them if the distribution of load in the system has changed. Since only one virtual node is active, the overhead for neighbor information storage and neighbor maintenance will not be increased in a real node. As in the Chord with the original virtual server method, this scheme gives each real node a small number of addresses on the Chord ring, preserving Chord's protection against address spoofing by malicious nodes trying to disrupt the routing layer. Combining the virtual node activation load-balancing scheme with the Koorde routing protocol (Kaashoek, 2003), the authors got a protocol that simultaneously offers (i) O(log n) degree per real node, (ii) O(log n/log log n) lookup hops, and (iii) constant factor load balance. The authors claimed that previous protocols could achieve any two of these but not all three. Generally speaking, achieving (iii) required operating O(log n) virtual nodes, which pushed the degree to $O(\log^2 n)$ and failed to achieve (i).

3.1.2 ID Assignment or Reassignment

In this category of load balancing methods, most proposals are similar in that they consider a number of (typically, Θ(log n)) locations for a node and select the one which gives the best load balance. The proposals differ in which locations should be considered, and when the selection should be conducted (Godfrey, 2005). Some proposals arrange a newly-jointed node to select a location, while others let nodes re-select a location when a node is overloaded.

Naor and Weider (2003) proposed a method in which a node checks Θ(log n) random IDs when joining, and chooses the ID which leads to the best load balance. They show that this method produces a maximum share of 2 if there are no node deletions. Share is an important metric for evaluating the performance of a load balancing method (Godfrey, 2005). Node v's share is defined as:

$$share\ (v) = \frac{f_v}{c_v / n},$$

where f_v is the ID space assigned to node v, and c_v is the normalized capacity of node v such that the average capacity is 1 and $\sum_v c_v = n$. To handle load imbalance incurred by node departures, nodes are divided into groups of Θ(log n) nodes and periodically reposition themselves in each group. Adler *et al.* (Adler, 2003) proposed to let a joining node randomly contacts an existing node already in the DHT. The joining node then chooses an ID in the longest interval owned by one of the proposed node's O(log n) neighbors to divide the interval into half. As a result, the intervals owned by nodes have almost the same length, leading to an O(1) maximum share.

Manku (Manku, 2004) proposed a load balancing algorithm. In the algorithm, a newly-joined node randomly chooses a node and splits in half the largest interval owned by one of the Θ(log n) nodes adjacent to the chosen node in the ID space. This achieves a maximum share of 2 while moving at most one node ID for each node arrival or departure. It extends to balancing within a factor 1 + ε but moves Θ (1/ε) IDs for any ε>0. As mentioned that Karger and Ruhl (Karger, 2004) proposed an algorithm in which each node has O(log n) virtual nodes, i.e. IDs, and periodically selects an ID among them as an active ID. This has maximum share 2 + ε, but requires reassignment of O(log log n) IDs per arrival or departure.

Bienkowski *et al.* (2005) proposed a node departure and re-join strategy to balance the key ID intervals across the nodes. In the algorithm, lightly loaded nodes leave the system and rejoin the system with a new ID to share the load of heavily loaded ones. The strategy reduces the number of reassignments to a constant, but shows only O(1) maximum share.

Byers *et al.* (Byers, 2003) proposed the use of the "power of two choices" algorithm. In this algorithm, each object is hashed to $d \geq 2$ different IDs, and is placed in the least loaded node of the nodes responsible for those IDs. The other nodes are given a redirection pointer to the destination node so that searching is not slowed significantly. For homogeneous nodes and objects and a static system, picking $d = 2$ achieves a load balance within a (log log n) factor of optimal, and when $d = \Theta(\log n)$, the load balance is within a constant factor of optimal.

The ID assignment or reassignment methods reassign IDs to nodes in order to maintain the load balance when nodes arrive and depart the system. The object transfer and neighbor update involved in ID rearrangement would incur a high overhead. Moreover, few methods directly take into account the heterogeneity of file load.

3.1.3 Load Transfer

The virtual server methods and key assignment and reassignment methods ignore the heterogeneity of file load. Further load imbalance may result from non-uniform distribution of files in the identifier space and a high degree of heterogeneity in file loads and node capacities. In addition, few of the methods are able to deal with both the network churn and proximity. In general, the DHT churn should be dealt with by randomized matching between heavily loaded nodes with lightly loaded nodes. Load transfer methods to move load from heavily loaded nodes to lightly loaded nodes can deal with these problems.

Rao *et al.* (2003) proposed three algorithms to rearrange load based on nodes' different capacities: one-to-one, many-to-many, and one-to-many. Their basic idea is to move virtual servers, i.e. load, from heavily loaded nodes to lightly loaded nodes so that each node's load does not exceed its capacity. Specifically, the method periodically collects the information of servers' load status, which helps load rearrangement between heavily loaded nodes and lightly loaded nodes. The algorithms are different primarily in the amount of information used to decide load rearrangement. In the one-to-one algorithm, each lightly loaded server randomly probes nodes for a match with a heavily loaded one. In the many-to-many algorithm, each heavily loaded server sends its excess virtual nodes to a global pool, which executes load rearrangement periodically. The one-to-one scheme produces too many probes, while the many-to-many scheme increases overhead in load rearrangement. As a trade-off, the one-to-many algorithm works in a way that each heavily loaded server randomly chooses a directory which contains information about a number of lightly loaded servers, and moves its virtual servers to lightly loaded servers until it is not overloaded anymore.

In a DHT overlay, a node's load may vary greatly over time since the system can be expected to experience continuous insertions and deletions of objects, skewed object arrival patterns, and continuous arrival and departure of nodes. To cope with this problem, Godfrey *et al.* (2006) extended Rao's work (Rao, 2003) for dynamic DHT networks with rapid arrivals and departures of items and nodes. In their approach, if a node's capacity utilization exceeds a predetermined threshold, its excess virtual servers will be moved to a lightly loaded node immediately without waiting for the next periodic balancing. This work studied this algorithm by using extensive simulations over a wide set of system scenarios and algorithm parameters.

Most recently, Karger and Ruhl (2004) proved that the virtual server method could not be guaranteed to handle item distributions where a key ID interval has more than a certain fraction of the load. As a remedy, they proposed two schemes with provable features: moving items and moving nodes to achieve equal load between a pair of nodes, and then achieves a system-wide load balance state. In the moving items scheme, every node occasionally contacts a random other node. If one of the two nodes has much larger load than the other, then items are moved from the heavily loaded node to the lightly loaded node until their loads become equal. In the moving nodes scheme, if a pair of nodes has very uneven loads, the load of the heavier node gets split between the two nodes by changing their addresses. However, this scheme breaks DHT mapping and cannot support key locations as usual. Karger and Ruhl (2004) provided a theoretic treatment for load balancing problem and proved that good load balance can be achieved by moving items if the fraction of address space covered by every node is O(1/n) (Karger, 2004).

Almost all of these algorithms assume the objective of minimizing the amount of moved load. The algorithms treat all nodes equally in random probing, and neglect the factor of physical proximity on the effectiveness of load balancing. With proximity consideration, load transferring and communication should be within physically close heavy and light nodes.

One of the first works to utilize the proximity information to guide load balancing is due to Zhu and Hu (2005). They presented a proximity-aware algorithm to take into account the node proximity information in load balancing. The authors suggested to build a K-nary tree (KT) structure on top of a DHT overlay. Each KT node is planted in a virtual server. A K-nary tree node reports the load information of its real server to its parent, until the tree root is reached. The root then disseminates final information to all the virtual nodes. Using this information, each real server can determine whether it is heavily loaded or not. Lightly loaded and heavily loaded nodes report their free capacity, excess virtual nodes information to their KT leaf nodes respectively. The leaf nodes will propagate the information upwards along the tree. When the total length of information reaches a certain threshold, the KT node would execute load rearrangement between heavily loaded nodes and lightly loaded nodes. The KT structure helps to use proximity information to move load between physically close heavily and lightly loaded nodes. However, the construction and maintenance of KT are costly, especially in churn. In churn, a KT will be destroyed without timely fixes, degrading load balancing efficiency. For example, when a parent fails or leaves, the load imbalance of its children in the subtree cannot be resolved before its recovery. Therefore, although the network is self-organized, the algorithm is hardly applicable to DHTs with churn. Besides, the tree needs to be reconstructed every time after virtual server transferring, which is imperative in load balancing. Second, a real server cannot start determining its load condition until the tree root gets the accumulated information from all nodes. This centralized process is inefficient and hinder the scalability improvement of P2P systems.

3.2 Case Study: Locality-Aware Randomized Load Balancing Algorithms

This section presents Locality-Aware Randomized load balancing algorithms (LAR) (Shen, 2007) that take into account proximity information in load balancing and deal with network dynamism meanwhile.

The algorithms take advantage of the proximity information of the DHTs in node probing and distribute application load among the nodes according to their capacities. The LAR algorithms introduce a factor of randomness in the probing of lightly loaded nodes in a range of proximity so as to make the probing process robust in DHTs with churn. The LAR algorithms further improve the efficiency by allowing the probing of multiple candidates at a time. Such a probing process is referred as d-way probing, $d \geq 1$. The algorithms are implemented in Cycloid (Shen, 2006), based on a concept of "moving item" (Karger, 2004) for retaining DHT network efficiency and scalability. The algorithms are also suitable for virtual server methods. The performance of the LAR load balancing algorithms is evaluated via comprehensive simulations. Simulation results demonstrate the superiority of a locality-aware 2-way randomized load balancing algorithm, in comparison with other pure random approaches and locality-aware sequential algorithms. In DHTs with churn, it performs no worse than the best churn resilient algorithm. In the following, Cycloid DHT is first introduced before the LAR algorithms are presented.

3.2.1 Cycloid: A Constant-Degree DHT

Cycloid (Shen, 2006) is a lookup efficient constant-degree DHT that we recently proposed. In a Cycloid system with $n = d \cdot 2d$ nodes, each lookup takes $O(d)$ hops with $O(1)$ neighbors per node. In this section, we give a brief overview of the Cycloid architecture and its self-organization mechanism, focusing on the structural features related to load balancing.

ID and Structure. In Cycloid, each node is represented by a pair of indices $(k, a_{d-1} a_{d-2} \ldots a_0)$, where k is a cyclic index and $a_{d-1} a_{d-2} \ldots a_0$ is a cubical index. The cyclic index is an integer, ranging from 0 to $d - 1$ and the cubical index is a binary number between 0 and $2^d - 1$. Each node keeps a routing table and two leaf sets, inside leaf set and outside leaf set, with a total of 7 entries to maintain its connectivity to the rest of the system. Table 1 shows a routing state table for node (4,10111010) in an 8-dimensional Cycloid, where x indicates an arbitrary binary value. Its corresponding links in both cubical and cyclic aspects are shown in Figure 1. In general, a node (k, a_{d-1} a_{d-2} . . . a_0), k 6= 0, has one cubical neighbor (k − 1, $a_{d-1}a_{d-2}$ . . . a_k xx...x) where x denotes an arbitrary bit value, and two cyclic neighbors (k−1, b_{d-1} b_{d-2} . . . b_0) and (k−1, c_{d-1} c_{d-2} . . . c_0). The cyclic neighbors are the first larger and smaller nodes with cyclic index k−1 mod d and their most significant different bit with the current node in cubical indices is no larger than (k − 1).

That is, (k-1, $b_{d-1} \ldots b_1b_0$) = min{∀(k-1, $y_{d-1} \ldots y_1y_0$)|$y_{d-1} \ldots y_0 \geq a_{d-1} \ldots a_1a_0$},

(k-1, $c_{d-1} \ldots c_1c_0$) = max{∀(k-1, $y_{d-1} \ldots y_1y_0$)|$y_{d-1} \ldots y_0 \leq a_{d-1} \ldots a_1a_0$}.

Table 1. Routing table of a cycloid node (4,101-1-1010).

NodeID (4,101-1-1010)	
Routing Table	
Cubical neighbor: (3,101-0-xxxx)	
Cyclic neighbor: (3,101-1-1100)	
Cyclic neighbor: (3,101-1-0011)	
Leaf Sets (half smaller, half larger)	
Inside Leaf Set	
(3,101-1-1010)	(6,101-1-1010)
Outside Leaf Set	
(7,101-1-1001)	(6,101-1-1011)

The node with a cyclic index k = 0 has no cubical neighbor or cyclic neighbors. The node with cubical index 0 has no small cyclic neighbor, and the node with cubical index $2^d - 1$ has no large cyclic neighbor. The nodes with the same cubical index are ordered by their cyclic index (mod d) on a local circle. The inside leaf set of a node points to the node's predecessor and successor in the local circle. The largest cyclic index node in a local circle is called the primary node of the circle. All local circles together form a global circle, ordered by their cubical index (mod 2^d). The outside leaf set of a node points to the primary nodes in its preceding and succeeding small circles in the global circle. The Cycloid connection pattern is resilient in the sense that even if many nodes are absent, the remaining nodes are still capable of being connected. The Cycloid DHT assigns keys onto its ID space by the use of a consistent hashing function. For a given key, the cyclic index of its mapped node is set to its hash value modulated by d and the cubical index is set to the hash value divided by d. If the target node of an item key (k, $a_{d-1} \ldots a_1 a_0$) is not present in the system, the key is assigned to the node whose ID is first numerically closest to $a_{d-1}\ a_{d-2} \ldots a_0$ and then numerically closest to k.

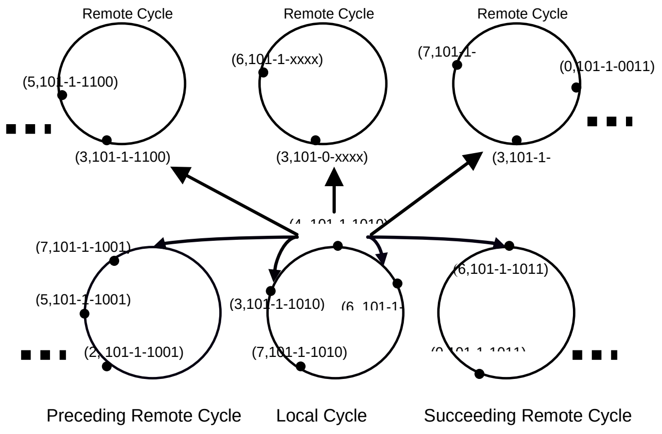


Figure 1. Cycloid node routing links state.

Self-organization. P2P systems are dynamic in the sense that nodes are frequently joining and departing from the network. Cycloid deals with the dynamism in a distributed manner. When a new node joins, it initializes its routing table and leaf sets, and notifies the nodes in its inside leaf set of its participation. It also needs to notify the nodes in its outside leaf set if it becomes the primary node of its local circle. Before a node leaves, it notifies its inside leaf set nodes, as well. Because a Cycloid node has no incoming connections for cubical and cyclic neighbors, a leaving node cannot notify those who take it as their cubical neighbor or cyclic neighbor. The need to notify the nodes in its outside leaf set depends on whether the leaving node is a primary node or not. Updating cubical and cyclic neighbors are the responsibility of system stabilization, as in Chord.

3.2.2 Load Balancing Framework

This section presents a framework for load balancing based on item movement on Cycloid. It takes advantage of the Cycloid's topological properties and conducts a load balancing operation in two steps: local load balancing within a local circle and global load balancing between circles. A general approach with consideration of node heterogeneity is to partition the nodes into a super node with high capacity and a class of regular nodes with low capacity (Fasttrack, 2001; Yang, 2003). Each super node, together with a group of regular nodes, forms a cluster in which the super node operates as a server to the others. All the super nodes operate as equals in a network of super-peers. Super-peer networks strike a balance between the inherent efficiency of centralization and distribution, and take advantage of capacity heterogeneity, as well. Recall that each local circle in Cycloid has a primary node. We regard Cycloid as a quasi-super-peer network by assigning each primary node as a leading super node in its circle. A node is designated as supernode if its capacity is higher than a pre-defined threshold. The Cycloid rules are modified for node join and leave slightly to ensure that every primary node meets the capacity requirement of supernodes. If the cyclic ID selected by a regular node is the largest in its local circle, it needs to have another choose unless it is the bootstrap node of the circle. In the case of primary node departure or failure, a supernode needs to be searched in the primary node's place if the node with the second largest cyclic ID in the circle is not a super node. This operation can be regarded as the new supernode leaves and re-joins the system with the ID of the leaving or failing primary node. Let $L_{i,k}$ denote the load of item k in node i. It is determined by the item size $S_{i,k}$ and the number of visits of the item $V_{i,k}$ during a certain time period. That is, $L_{i,k} = S_{i,k} \times V_{i,k}$. The actual load of a real server i, denoted by L_i, is the total load of all of its items:

$$L_i = \sum_{k=1}^{m_i} L_{i,k} ,$$

assuming the node has m_i items. Let C_i denote the capacity of node i; it is defined as a pre-set target load which the node is willing to hold. We refer to the node whose actual load is no larger than its target load (i.e. $L_i \le C_i$) as a light node; otherwise a heavy one. We define utilization of a node i, denoted by NU_i, as the fraction of its target capacity that is occupied. That is, $NU_i = L_i/C_i$. System utilization, denoted by SU, is the ratio of the total actual load to the total node capacity. Each node contains a list of data items, labeled as D_k, k = 1, 2, ... To make full use of node capacity, the excess items chosen to transfer should be with minimum load. We define excess items of a heavy node as a subset of the resident items, satisfying the following condition. Without loss of generality, we assume the excess items are $\{D_1, D_2, . . . , D_{m'}\}$, $1 \le m' \le m$. Their corresponding loads are $\{L_{i,1}, ..., L_{i,m'}\}$. The set of excess items is determined in such a way that

$$\text{minimizes} \quad \sum_{k=1}^{m'} L_{i,k} \qquad (1)$$

$$\text{subject to} \quad (L_i - \sum_{k=1}^{m'} L_{i,k}) \le C_i \qquad (2)$$

Each primary node has a pair of sorted donating and starving lists which store the load information of all nodes in its local cycle. A donating sorted list (DSL) is used to store load information of light nodes and a starving sorted list (SSL) is used to store load information of heavy nodes as shown in Table 2. The free capacity of light node i is defined as $\delta L_i = C_i - L_i$. Load information of heavy node I includes the information of its excess items in a set of 3-tuple representation: $< L_{i,1}, D_{i,1}, A_i >$,$< L_{i,k}, D_{i,k}, A_i >$, . . . ,$<L_{i,m'}, D_{i,m'}, A_i >$, in which A_i denotes the IP address of node i. Load information of light node j is represented in the form of $< \delta Lj ,Aj >$. An SSL is sorted in a descending order of Li,k; minLi,k represents the item with the minimum load in the primary node's starving list. A DSL is sorted in an ascending order of δLj ; max δLj represents the maximum δLj in the primary node's donating list. Load rearrangement is executed between a pair of DSL and SSL, as shown in Algorithm 1.

Table 2. Donating and starving sorted lists.

Load information in a primary node	
Donating sorted list	Starving sorted list
$< \delta L_j , A_j >$	$< L_{i,1}, D_{i,1}, A_i >$
…	…
$< \delta L_m , A_m >$	$< L_{i,k}, D_{i,k}, A_i >$

This scheme guarantees that heavier items have a higher priority to be reassigned to a light node, which means faster convergence to a system-wide load balance state. A heavy item $L_{i,k}$ is assigned to the most-fit light node with δLj which has minimum free capacity left after the heavy item $L_{i,k}$ is transferred to it. It makes full use of the available capacity. Our load balancing framework is based on item movement, which transfers items directly instead of virtual servers to save cost. Cycloid maintains two pointers for each transferred item. When an item D is transferred from heavy node i to light node j, node i will have a forward pointer in D location pointing to the item D in j's place; item D will have a backward pointer to node i indicating its original host. When queries for item D reach node i, they will be redirected to node j with the help of forward pointer. If item D needs to be transferred from node j to another node, say g, for load balancing, node j will notify node i via its backward pointer of the item's new location.

Algorithm 1: Primary node periodically performs load rearrangement between a pair of DSL and SSL

```
for each item k in SSL do
  for each item j in DSL do
    if $L_{i,k} \leq \delta L_j$ then
      Item k is arranged to be transferred from i to j
      if $\delta L_j - L_{i,k} > 0$ then
        Put <( $\delta L_i - L_{i,k}$),$A_i$> back to DSL
```

We use a centralized method in local load balancing, and a decentralized method in global load balancing. Each node (k, $a_{d-1}a_{d-2} \ldots a_0$) periodically reports its load information to the primary node in its local circle. Unlike a real super-peer network, Cycloid has no direct link between a node and the primary node. The load information needs to be forwarded using Cycloid routing algorithm, which ensures the information reaches the up-to-the-minute primary node. Specifically, the information is targeted to the node ($d-1$, $a_{d-1}a_{d-2} \ldots a_0$). By the routing algorithm, the destination it reaches, say node i, may be the primary node or its successor depending on which one is closer to the ID. If the cyclic index of the successor(i) is larger than the cyclic index of i, then the load information is forwarded to the predecessor(i), which is the primary node. Otherwise, node i is the primary node. According to the Cycloid routing algorithm, each report needs to take d/2 steps in the worst case. Cycloid cycle contains a primary node all the time. Since the load information is guaranteed to reach the up-to-the-minute primary node, there is no serious advert effect of primary node updates on load balancing. After receiving the load information, the primary node puts it to its own DSL and SSL accordingly. A primary node with nonempty starving list (PNS) first performs local load rearrangement between its DSL and

SSL. Afterwards, if its SSL is still not empty, it probes other primary nodes' DSLs for global load rearrangement one by one until its SSL becomes empty. When a primary node does't have enough capacity for load balancing, it can search for a high capacity node to replace itself. We arrange the PNS to initiate probing because the probing process will stop once it is not overloaded. If a node of nonempty donating list initiates probing, the probing process could proceed infinitely, incurring much more communication messages and bandwidth cost. Because primary nodes are super peers with high capacities, they are less likely to be overloaded in the load balancing. This avoids the situation that heavy nodes will be overloaded if they perform probing, such as in the schemes in (Rao, 2003). This scheme can be extended to perform load rearrangement between one SSL and multiple DSLs for improvement.

3.2.3 Locality-Aware Randomized Load Balancing Algorithms

The load balancing framework in the preceding section facilitates the development of load balancing algorithms with different characteristics. A key difference between the algorithms is, for a PNS, how to choose another primary node for a global load rearrangement between their SSL and DSL. It affects the efficiency and overhead to reach a system-wide load balance state.

D-way Randomized Probing. A general approach to dealing with the churn of DHTs is randomized probing. In the policy, each PNS probes other primary nodes randomly for load rearrangement. A simple form is one-way probing, in which a PNS, say node i, probes other primary nodes one by one to execute load rearrangement SSL_i and DSL_j, where j is a probed node. We generalize the one-way randomized probing policy to a d-way probing, in which d primary nodes are probed at a time, and the primary node with the most total free capacity in its DSL is chosen for load rearrangement. A critical performance issue is the choice of an appropriate value d. The randomized probing in our load balancing framework is similar to load balancing problem in other contexts: competitive online load balancing and supermarket model. Competitive online load balancing is to assign each task to a server on-line with the objective of minimizing the maximum load on any server, given a set of servers and a sequence of task arrivals and departures. Azar et al. (1994) proved that in competitive online load balancing, allowing each task to have two server choices to choose a less loaded server instead of just one choice can exponentially minimize the maximum server load and result in a more balanced load distribution. Supermarket model is to allocate each randomly incoming task modeled as a customer with service requirements, to a processor (or server) with the objective of reducing the time each customer spends in the system. Mitzenmacher et al. (1997) proved that allowing a task two server choices and to be served at the server with less workload instead of just one choice leads to exponential improvements in the expected execution time of each task. But a poll size larger than two gains much less substantial extra improvement. The randomized probing between the lists of SSLs and DSLs is similar to the above competitive load balancing and supermarket models if we regard SSLs as tasks, and DSLs as servers. But the random probing in P2P systems has a general workload and server models. Servers are dynamically composed with new ones joining and existent ones leaving. Servers are heterogeneous with respect to their capacities. Tasks are of different sizes and arrive in different rates. In (Fu, 2008), we proved the random probing is equivalent to a generalized supermarket model and showed the following results.

Theorem 5.1: Assume servers join in a Poisson distribution. For any fixed time interval $[0,T]$, the length of the longest queue in the supermarket model with $d = 1$ is $\ln n / \ln \ln n(1+O(1))$ with high probability; the length of the longest queue in the model with $d \geq 2$ is $\ln \ln n / \ln d + O(1)$, where n is the number of servers.

The theorem implies that 2-way probing could achieve a more balanced load distribution with faster speed even in churn, because 2-way probing has higher possibility to reach an active node than 1-way probing, but d-way probing, d > 2, may not result in much additional improvement.

Locality-Aware Probing. One goal of load balancing is to effectively keep each node lightly loaded with minimum load balancing overhead. Proximity is one of the most important performance factors. Mismatch between logical proximity abstraction and physical proximity information in reality is a big obstacle for the deployment and performance optimization issues for P2P applications. Techniques to exploit topology information in overlay routing include geographic layout, proximity routing and proximity-neighbor selection (Castro, 2002).

The proximity-neighbor selection and topologically-aware overlay construction techniques in (Xu, 2003; Castro, 2002; Waldvogel, 2002) are integrated into Cycloid to build a topology-aware Cycloid. As a result, the topology-aware connectivity of Cycloid ensures that a message reaches its destination with minimal overhead. Details of topology-aware Cycloid construction will be presented in Section 3.2.4.

In a topology-aware Cycloid network, the cost for communication and load movement can be reduced if a primary node contacts other primary nodes in its routing table or primary nodes of its neighbors. In general, the primary nodes of a node's neighbors are closer to the node than randomly chosen primary nodes in the entire network, such that load is moved between closer nodes. This method should be the first work that handles the load balancing issue with the information used for achieving efficient routing. There are two methods for locality-aware probing: randomized and sequential method.

1. *Locality-aware randomized probing (LAR).* In LAR, each PNS contacts primary nodes in a random order in its routing table or primary nodes of its neighbors except the nodes in its inside leaf set. After all these primary nodes have been tried, if the PNS's SSL is still nonempty, global random probing is started in the entire ID space.
2. *Locality-aware sequential probing (Lseq).* In Lseq, each PNS contacts its larger outside leaf set *Successor(PNS)*. After load rearrangement, if its SSL is still nonempty, the larger outside leaf set of *Successor(PNS)*, *Successor(Successor(PNS))* is tried. This process is repeated, until that SSL becomes empty. The distances between a node and its sequential nodes are usually smaller than distances between the node and randomly chosen nodes in the entire ID space.

3.2.4 Performance Evaluation

We designed and implemented a simulator in Java for evaluation of the load balancing algorithms on topology-aware Cycloid. Table 3 lists the parameters of the simulation and their default values. The simulation model and parameter settings are not necessarily representative of real DHT applications. They are set in a similar way to related studies in literature for fair comparison. We will compare the different load balancing algorithms in Cycloid without churn in terms of the following performance metrics; the algorithms in Cycloid with churn will also be evaluated.

1. *Load movement factor*, defined as the total load transferred due to load balancing divided by the system actual load, which is system target capacity times SU. It represents load movement cost.

2. *Total time of probings*, defined as the time spent for primary node probing assuming that probing one node takes 1 time unit, and probing a number of nodes simultaneously also takes 1 time unit. It represents the speed of probing phrase in load balancing to achieve a system-wide load balance state.
3. *Total number of load rearrangements*, defined as the total number of load rearrangement between a pair of SSL and DSL. It represents the efficiency of probing for light nodes.
4. *Total probing bandwidth*, defined as the sum of the bandwidth consumed by all probing operations. The bandwidth of a probing operation is the sum of bandwidth of all involved communications, each of which is the product of the message size and physical path length of the message traveled. It is assumed that the size of a message asking and replying for information is 1 unit. It represents the traffic burden caused by probings.
5. *Moved load distribution*, defined as the cumulative distribution function (CDF) of the percentage of moved load versus moving distance. It represents the load movement cost for load balance. The more load moved along the shorter distances, the less load balancing costs.

Topology-aware Cycloid Construction. GT-ITM (transit-stub and tiers) (Zegura, 1996) is a network topology generator, widely used for the construction of topology-aware overlay networks (Ratnasamy, 2002; Xu, 2003; Xu, 2003; Gummadi, 2003). We used GT-ITM to generate transit-stub topologies for Cycloid, and get physical hop distance for each pair of Cycloid nodes. Recall that we use proximity-neighbor selection method to build topology-aware Cycloid; that is, it selects the routing table entries pointing to the physically nearest among all nodes with nodeID in the desired portion of the ID space.

Table 3. Simulation settings and algorithm parameters.

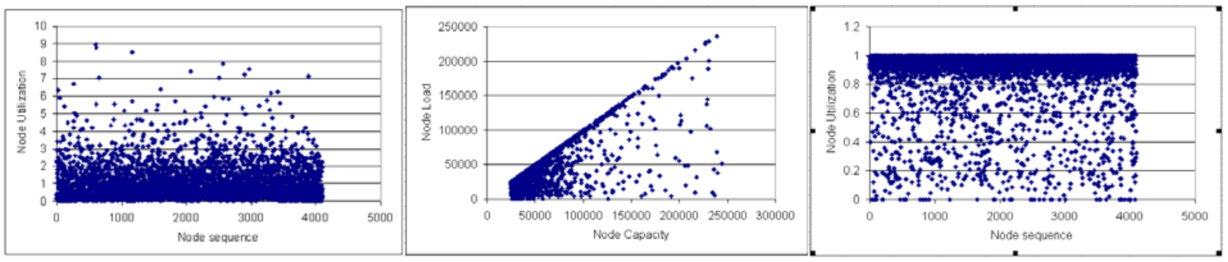
Environmental Parameter	**Default Value**
Object arrival location	Uniform over ID space
Number of nodes	4906
Node capacity	Bounded Pareto: shape 2 lower bound: 2500, upper bound: 2500 * 10
Number of items	20480
Existing item load	Bounded Pareto: shape 2 lower bound: mean item actual load / 2 upper bound: mean item actual load / 2 * 10

We use landmark clustering and Hilbert number (Xu, 2003) to cluster Cycloid nodes. Landmark clustering is based on the intuition that close nodes are likely to have similar distances to a few landmark nodes. Hilbert number can convert *d* dimensional landmark vector of each node to one dimensional index while still preserve the closeness of nodes. We selected 15 nodes as landmark nodes to generate the landmark vector and a Hilbert number for each node cubic ID. Because the nodes in a stub domain have close (or even same) Hilbert numbers, their cubic IDs are also close to each other. As a result, physically close nodes are close to each other in the DHT's ID space, and nodes in one cycle are physically close to each other. For example, assume nodes i and j are very close to each other in physical locations but far away from node *m*. Nodes i and j will get approximately equivalent landmark vectors, which are different from m's. As a result, nodes i and j would get the same cubical IDs and be assigned to the circle different from m's. In the landmark approach, for each topology, we choose landmarks at random with the only condition that the landmarks are separated from each other by four hops. More sophisticated placement schemes, as described in (Jamin, 2000) would only serve to improve our results.

Our experiments are built on two transit-stub topologies: “ts5k-large” and “ts5k-small” with approximately 5,000 nodes each. In the topologies, nodes are organized into logical domains. We classify the domains into two types: transit domains and stub domains. Nodes in a stub domain are typically an endpoint in a network flow; nodes in transit domains are typically intermediate in a network flow. “ts5k-large” has 5 transit domains, 3 transit nodes per transit domain, 5 stub domains attached to each transit node, and 60 nodes in each stub domain on average. “ts5k-small” has 120 transit domains, 5 transit nodes per transit domain, 4 stub domains attached to each transit node, and 2 nodes in each stub domain on average. “ts5k-large” has a larger backbone and sparser edge network (stub) than “ts5k-small”. “ts5k-large” is used to represent a situation in which Cycloid overlay consists of nodes from several big stub domains, while “ts5k-small” represents a situation in which Cycloid overlay consists of nodes scattered in the entire Internet and only few nodes from the same edge network join the overlay. To account for the fact that interdomain routes have higher latency, each interdomain hop counts as 3 hops of units of latency while each intradomain hop counts as 1 hop of unit of latency.

Effectiveness of LAR Algorithms. In this section, we will show the effectiveness of LAR load balancing algorithm. First, we present the impact of LAR algorithm on the alignment of the skews in load distribution and node capacity when the system is fully loaded. Figure 2(a) shows the initial node utilization of each node. Recall that node utilization is a ratio of the actual load to its target (desired) load. Many of the nodes were overloaded before load balancing.

Load balancing operations drove all node utilizations down below 1 by transferring excess items between the nodes, as shown in Figure 2(b). Figure 2(c) shows the scatterplot of loads according to node capacity. It confirms that the capacity-aware load balancing feature of the LAR algorithm. Recall that LAR algorithm was based on item movement, using forward



(a) Before load balancing (b) After load balancing (c) Node load after balancing

Figure 2. Effect of load balancing.

pointers to keep DHT lookup protocol. We calculated the fraction of items that are pointed to by forward pointers in systems of different utilization levels. We found that the fraction increased linearly with the system load, but it would be no higher than 45% even when the system becomes fully loaded. The cost is reasonably low compared to the extra space, maintenance cost and efficiency degradation in virtual server load balancing approach.

We measured the load movement factors due to different load balancing algorithms: one-way random (R_1), two-way random (R_2), LAR_1, LAR_2, and Lseq, on systems of different loads and found that the algorithms led to almost the same amount of load movement in total at any given utilization level. This is consistent with the observations by Rao et al. (2003) that the load moved depends only on distribution of loads, the target to be achieved, but not on load balancing algorithms. This result suggests that an effective load balancing algorithm should explore to move the same amount of load along shorter distance and in shorter time to reduce load balancing overhead.

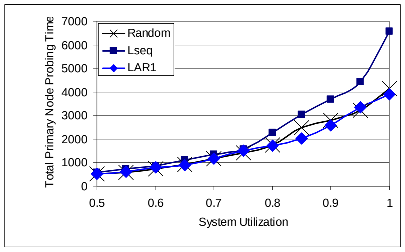
In the following, we will examine the performance of various load balancing algorithms in terms of other performance metrics. Because metrics (2) and (3) are not affected by topology, the results of them in "ts5k-small" will not be presented sometimes.

Comparison with Other Algorithms. Figure 3(a) shows the probing process in Lseq takes much more time than R_1 and LAR_1. This implies that random algorithm is better than sequential algorithm in probing efficiency.

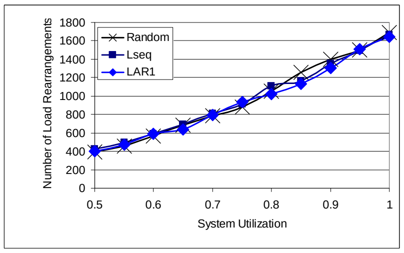
Figure 3(b) shows that the numbers of rearrangements of the three algorithms are almost the same. This implies that they need almost the same number of load rearrangement to achieve load balance. However, long probing time of Lseq suggests that it is not as efficient as random probing. It is consistent with the observation of Mitzenmacher in (Mitzenmacher, 1997) that simple randomized load balancing schemes can balance load effectively. Figure 3(c) and (d) show the performance of the algorithms in "ts5k-large". From Figure 3(c), we can observe that unlike in lightly loaded systems, in heavily loaded systems, R_1 takes more bandwidth than LAR_1 and Lseq, and the performance gap increases as the system load increases. This is because that much less probings are needed in a lightly loaded system, causing less effect of probing distance on bandwidth consumption.

The bandwidth results of LAR and Lseq are almost the same when the SU is under 90%; when the SU goes beyond 0.9, LAR consumes more bandwidth than Lseq. This is due to the fact that in a more heavily loaded system, more nodes need to be probed in the entire ID space, leading to longer load transfer distances. Figure 3(d) shows the moved load distribution in load balancing as the SU approaches 1. We can see that LAR_1 and Lseq are able to transfer about 60% of global moved load within 10 hops, while R_1 transfers only about 15% because R_1 is locality-oblivious.

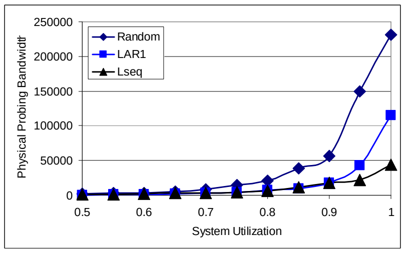
Figure 3(e) and (f) show the performance of the algorithms in "ts5k-small". These results also confirm that LAR_1 achieve better locality-aware performance than R_1, although the improvement is not so significant as in "ts5k-large". It is because that in "ts5k-small" topology,



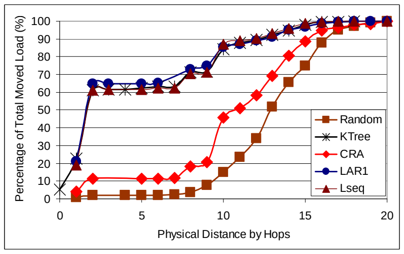
(a) Total primary node probing in “ts5k-large”



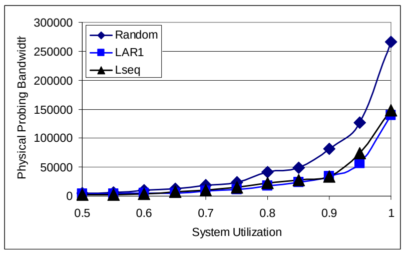
(b) Total number of load in “ts5k-large”



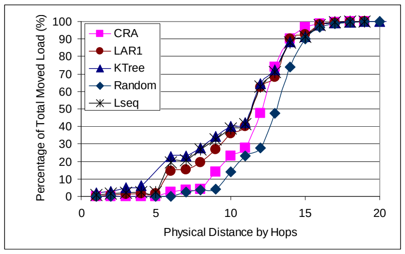
(c) Total bandwidth of probings in “ts5k-large”



(d) CDF of moved load distribution in “ts5klarge”



(e) Total bandwidth of probings in “ts5k-small”



(f) CDF of moved load distribution in “ts5ksmall”

Figure 3. Effect of load balancing due to different probing algorithms.

nodes are scattered in the entire network, and the neighbors of a primary node may not be physically closer than other nodes.

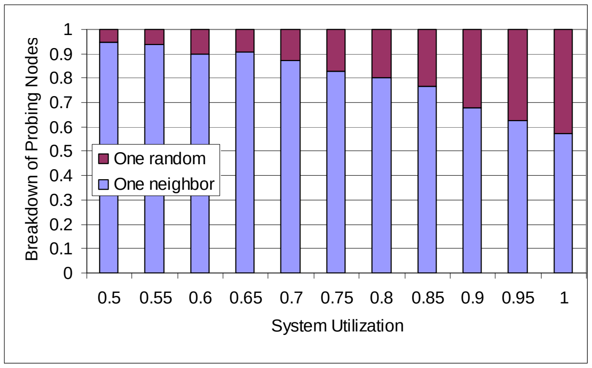
Figures 3(d) and (f) also include the results due to two other popular load balancing approaches: proximity-aware Knary Tree (KTree) algorithm (Zhu, 2005) and churn resilient algorithm (CRA) (Godfrey, 2006) for comparison. From the figures, we can see that LAR performs as well as KTree, and outperforms proximity-oblivious CRA, especially in “ts5k-large”. The performance gap between proximity-aware and proximity-oblivious algorithms is not as large as in “ts5k-small”. It is because the nodes in “ts5k-small” are scattered in the entire Internet with less locality.

In summary, the results in Figure 3 suggest that the randomized algorithm is more efficient than the sequential algorithm in the probing process. The locality-aware approaches can effectively assign and transfer loads between neighboring nodes first, thereby reduce network traffic and improve load balancing efficiency. The LAR algorithm performs no worse than the proximity-aware KTree algorithm. In Section 3.2.5, we will show LAR works much better for DHTs with churn.

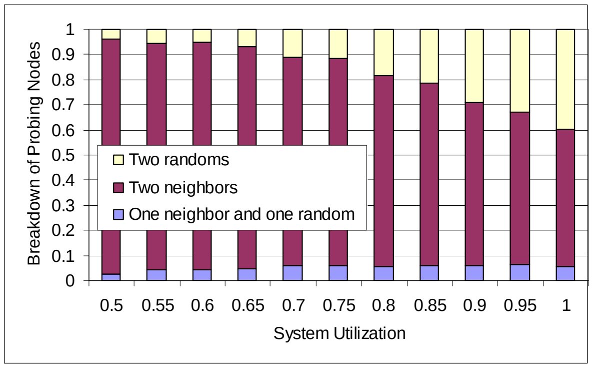
Effect of D-Way Random Probing. We tested the performance of the LAR_d algorithms with different probing concurrency degree d. Figure 5(a) shows that LAR_2 takes much less probing time than LAR_1. It implies that LAR_2 reduces the probing time of LAR_1 at the cost of more number of probings. Unlike LAR_1, in LAR_2, a probing node only sends its SSL to a node with more total free capacity in its DSL between two probed nodes. The more item transfers in one load rearrangement, the less probing time. It leads to less number of SSL sending operation of LAR_2 than LAR_1, resulting in less number of load rearrangements as shown in Figure 5(b). Therefore, simultaneous probings to get a node with more total free capacity in its DSL can save load balancing time and reduce network traffic load.

Figures 4(a) and (b) show the breakdown of total number of probed nodes in percentage that are from neighbors or randomly chosen in entire ID space in LAR_1 and LAR_2 respectively. Label “one neighbor and one random” represents the condition when there’s only one neighbor in routing table, then another probed node is chosen randomly from ID space. We can see that the percentage of neighbor primary node constitutes the most part, which means that neighbors can support most of system excess items in load balancing.

With SU increases, the percentage of neighbor primary node decreases because the neighbors’ DSLs don’t have enough free capacity for a larger number of excess items, then randomly chosen primary nodes must be resorted to. Figures 5(a) and (b) show that the probing efficiency of LAR_d (d>2) is almost the same as LAR_2, though they need to probe more nodes



(a) LAR$_1$

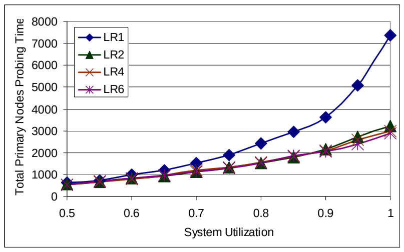


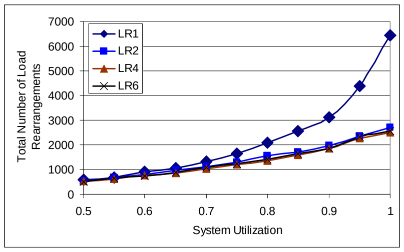
(b) LAR$_2$

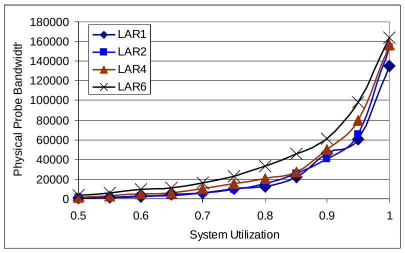
Figure 4. Breakdown of probed nodes.

than LAR$_2$. The results are consistent with the expectations in Section 3.2.1 that a two-way probing method leads to an exponential improvement over one-way probing, but a d-way (d>2) probing leads to much less substantial additional improvement. In the following, we will analyze whether the improvement of LAR$_d$ ($d \geq 2$) over LAR1 is at the cost of more bandwidth consumption or locality-aware performance degradation. We can observe from Figure 5(c) that the probing bandwidth of LAR$_2$ is almost the same as LAR$_1$.

Figure 5(d) shows the moved load distribution in global load balancing due to different algorithms. We can see that LAR$_2$ leads to an approximately identical distribution as LAR$_1$ and they cause slightly less global load movement cost than LAR$_4$ and LAR$_6$. This is because the more simultaneous probed nodes, the less possibility that the best primary node is a close neighbor node. These observations demonstrate that LAR$_2$ improves on LAR$_1$ at no cost of bandwidth consumption. It retains the advantage of locality-aware probing. Figures 5(e) and (f) show the performance of different algorithms in “ts5k-small”. Although the performance gap is not as wide as in ‘ts5k-large”, the relative performance between the algorithms retains.



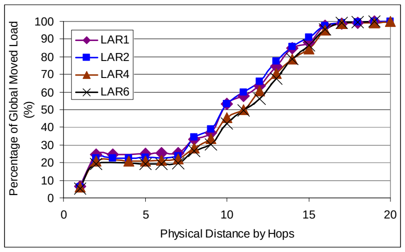


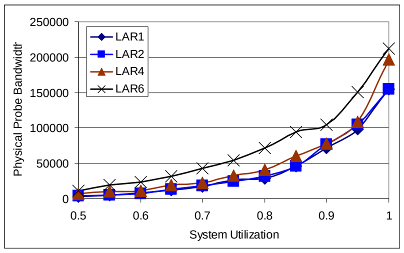


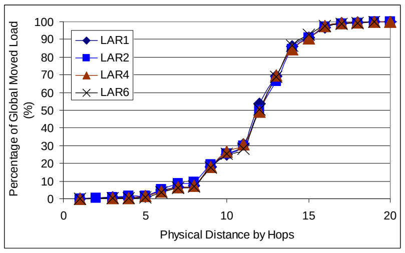
(a) Total primary node probing in "ts5k-large"

(b) Total number of load in "ts5k-large"

(c) Total bandwidth of probings in "ts5k-large"







(d) CDF of moved load distribution in "ts5klarge"

(e) Total bandwidth of probings in "ts5k-small"

(f) CDF of moved load distribution in "ts5ksmall"

Figure 5. Effect of load balancing due to different LAR algorithms.

In practice, nodes and items continuously join and leave P2P systems. It is hard to achieve the objective of load balance in networks with churn. We conducted a comprehensive evaluation of the LAR algorithm in dynamic situations and compare the algorithm with CRA, which was designed for DHTs with churn. The performance factors we considered include load balancing frequency, item arrival/departure rate, non-uniform item arrival pattern, and network scale and node capacity heterogeneity. We adopted the same metrics as in (Godfrey, 2006):

1. *The 99.9th percentile node utilization (99.9th NU)*. We measure the maximum 99.9th percentile of the node utilizations after each load balancing period T in simulation and take the average of these results over a period as the 99.9th NU. The 99.9th NU represents the efficiency of LAR to minimize load imbalance.

2. *Load moved/DHT load moved (L/DHT-L)*, defined as the total load moved incurred due to load balancing divided by the total load of items moved due to node joins and departures in the system. This metric represents the efficiency of LAR to minimize the amount of load moved.

Unless otherwise indicated, we run each trial of the simulation for 20T simulated seconds, where T is a parameterized load balancing period, and its default value was set to 60 seconds in our test. The item and node join/departure rates were modeled by Poisson processes. The default rate of item join/departure rate was 0.4; that is, there were one item join and one item departure every 2.5 seconds. We ranged node interarrival time from 10 to 90 seconds, with 10 second increment in each step. A node life time is computed by arrival rate times number of nodes in the system. The default system utilization SU was set to 0.8.

Performance Comparison With CRA in Churn. Figure 6 plots the performance due to LAR_1 and CRA versus node interarrival time during T period. By comparing results of LAR_1 and CRA, we can have a number of observations. First, the 99.9th NUs of LAR_1 and CRA are kept no more than 1 and 1.25 respectively. This implies that on average, LAR_1 is comparable with CRA in achieving the load balancing goal in churn. Second, LAR_1 moves up to 20% and CRA moves up to 45% of the system load to achieve load balance for SU as high as 80%. Third, the load moved due to load balancing is very small compared with the load moved due to node joins and departures and it is up to 40% for LAR1 and 53% for CRA. When the node interarrival time is 10, the L/DHT-L is the highest. It is because faster node joins and departures generate much higher load imbalance, such that more load transferred is needed to achieve load balance. The fact that the results of LAR_1 are comparable to CRA implies that LAR algorithm is as efficient as CRA to handle churn by moving a small amount load.

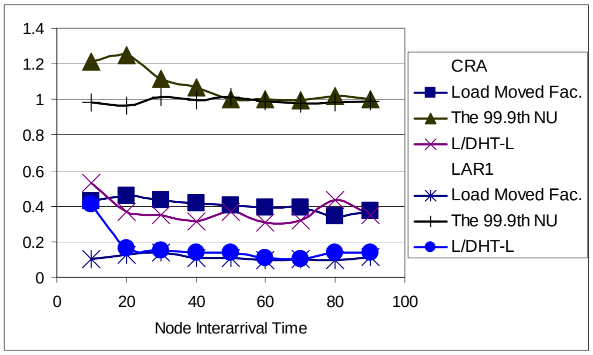
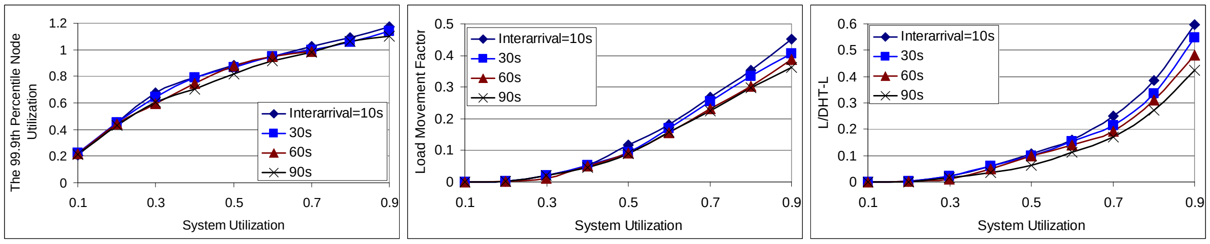


Figure 6. Effect of load balancing with churn.

The results in Figure 6 are due to a default node join/leave rate of 0.4. Figure 7 plots the 99.9th NU, load movement factor and the L/DHT-L as a function of SU with different node interarrival time respectively. We can observe that the results of the three metrics increase as SU increases. That’s because nodes are prone to being overloaded in a heavily loaded system, resulting in more load transferred to achieve load balance. We also can observe that the results of the metrics increase as interarrival time decreases, though they are not obvious. It is due to the fact that with faster node joins and departures, nodes are more easily to become overloaded, leading to the increase of the 99.9th NU and load moved in load balancing. Low NUs in different SU and node interarrival time means that the LAR is effective in maintaining each node light in a dynamic DHT with different node join/departure rate and different SUs, and confirms the churn-resilient feature of the LAR algorithm.

Impact of Load Balancing Frequency in Churn. It is known that high frequent load balancing ensures the system load balance at a high cost, and low frequent load balancing can hardly guarantee load balance at all time. In this simulation, we varied load balancing interval T from 60 to 600 seconds, at a step size of 60, and we conducted the test in a system with SU varies from 0.5 to 0.9 at a step size of 0.1.



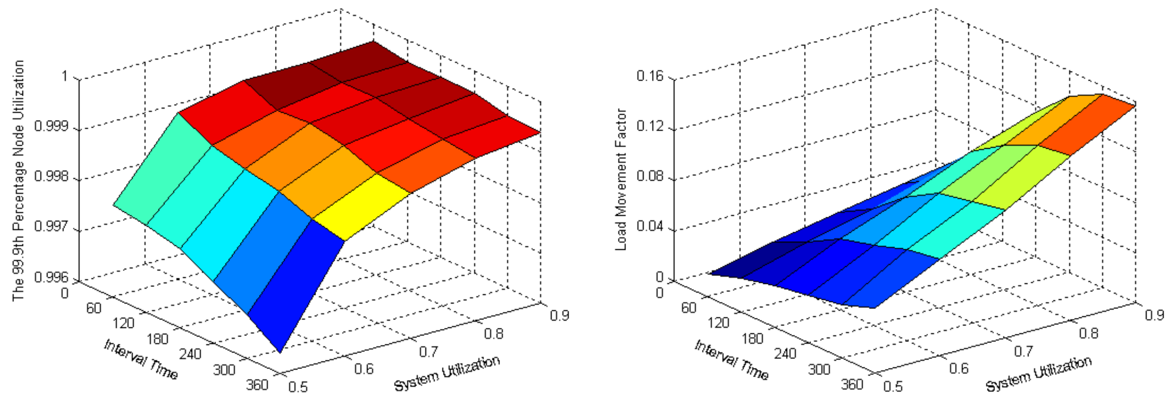
(a) The 99.9th percentile NU (b) Load movement factor (c) L/DHT-L

Fig. 7. Impact of system utilization under continual node joins and departures.

Figure 8(a) and (b) show the 99.9th NU and load movement factor in different system utilization and time interval. We can see that the 99.9th NU and load movement factor increase as SU increases. This is because that nodes are most likely to be overloaded in highly loaded system, leading to high maximum NU and a large amount of load needed to transfer for load balance.

Figure 8(a) shows that all the 99.9th NUs are less than 1, and when the actual load of a system consists more than 60% of its target load, the 99.9 NU quickly converges to 1. It implies that the LAR algorithm is effective in keeping every node light, and it can quickly transfer excess load of heavy nodes to light nodes even in a highly loaded system. Observing Figure 8(a) and (b), we find that in a certain SU, the more load moved, the lower 99.9th NU. It is consistent with our expectation that more load moved leads to move balanced load distribution.

Intuitively, a higher load balancing frequency should lead to less the 99.9th NU and more load moved. Our observation from Figure 8 is counter-intuitive. That is, the 99.9th NU increases and load movement factor decreases as load balancing is performed more frequently. Recall that the primary objective of load balancing is to keep each node not overloaded, instead of keeping the application load evenly distributed between the nodes. Whenever a node's utilization is

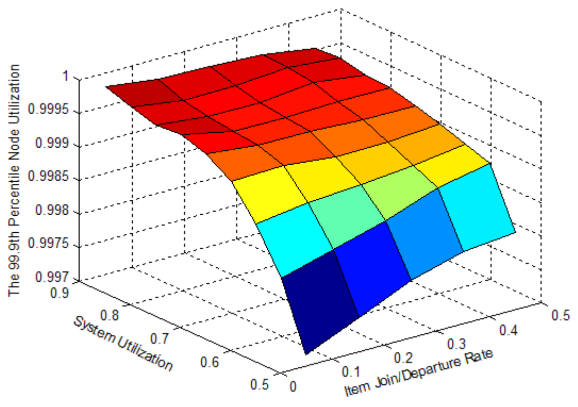


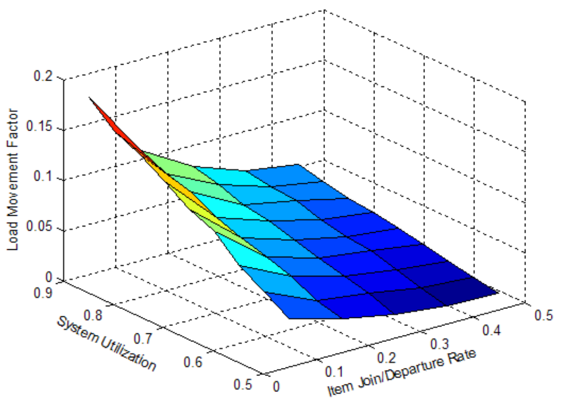
(a) The 99.9th percentage node utilization (b) Load movement factor

Figure 8. Impact of load balancing frequency.

below 1, it does not need to transfer its load to others. With a high load balancing frequency, few nodes are likely to be overloaded. They may have high utilizations less than 1, and end up with less load movement and high node utilization. Figure 8(b) reveals a linear relationship between the load movement factor and system utilization and that the slope of low frequency is larger than high frequency because of the impact of load balancing frequency on highly loaded systems.

Impact of Item Arrival/Departure Rate in Churn. Continuous and fast item arrivals increase the probability of overloaded nodes generation. Item departures generate nodes with available capacity for excess items. An efficient load balancing algorithm will find nodes with sufficient free capacity for excess items quickly in order to keep load balance state in churn. In this section, we evaluate the efficiency of LAR algorithm in the face of rapid item arrivals and departures. In this test, we varied item arrival/departure rate from 0.05 to 0.45 at a step size of 0.1, varied SU from 0.5 to 0.9 at a step size of 0.05, and measured the 99.9th NU and load movement factor in each condition. Figure 9(a) and (b), respectively, plot the 99.9th NU and load movement factor as functions of item arrival/departure rate. As expected, the 99.9th NU and





(a) The 99.9th percentage node utilization (b) Load movement factor

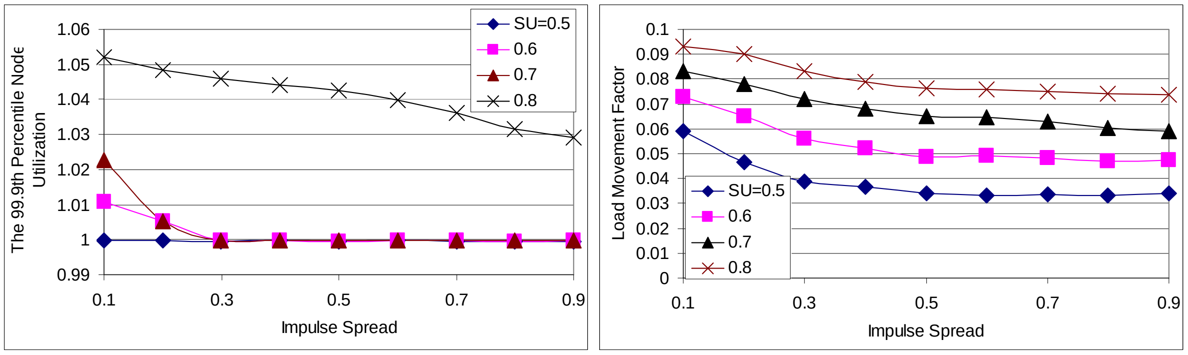
Figure 9. Impact of item arrival/departure rate.

load movement factor increase with system utilization. It is consistent with the results in the load balancing frequency test. Figure 9(a) shows that all the 99.9th NUs are less than 1, which means that the LAR is effective to assign excess items to light nodes in load balancing in rapid item arrivals and departures. From the figures, we can also see that when item arrival/departure rate increases, unlike in lightly loaded system, the 99.9th NU decreases in heavily loaded system. It is due to efficient LAR load balancing, in which more load rearrangements initiated timely by overloaded nodes with high item arrival rate. On the other hand, in the lightly loaded system, though the loads of nodes accumulate quickly with high item arrival rate, most nodes are still light with no need to move out load, leading to the increase of 99.9th NU. This is confirmed by the observation in Figure 9(b) that the load moved is higher in heavily loaded system than that in lightly loaded system, and movement factor drops faster in highly loaded system, which means that faster item departures lead to less load moved for load balance. Figure 9(b) demonstrates that the load movement factor drops as item arrival/departure rate increases. It is because that the total system load (denominator of load movement factor) grows quickly with a high item arrival/departure rate. In summary, item arrival/departure rate has direct effect on NU and load movement factor in load balancing, and LAR is effective to achieve load balance with rapid item arrivals and departures.

Impact of Non-uniform Item Arrivals in Churn. Furthermore, we tested LAR algorithm to see if it is churn-resilient enough to handle skewed load distribution. We define an “impulse” of items as a group of items that suddenly join in the system and their IDs are distributed over a contiguous interval of an ID space interval. We set their total load as 10% of the total system load, and varied the spread of interval from10% to 90% of the ID space.

Figure 10(a) shows that in different impulses and SUs, LAR algorithm kept the 99.9th NU less than 1.055, which implies that LAR algorithm can almost solve the impulses successfully. The 99.9th NU is high in high SU and low impulse spread. Except when SU equals to 0.8, the impulse with spread larger than 0.3 can be successfully resolved by LAR algorithm. When the impulse is assigned to a small ID space interval less than 0.3, the load of the nodes in that ID space interval accumulates quickly, leading to higher NUs. The situation becomes worse with higher SU, because there’s already less available capacity left in the system for the impulse. The curve of SU=0.8 is largely above others is mainly due to the item load and node capacity distributions, and the impulse load relative to the SU. In that case, it is hard to find nodes with large enough capacity to support excess items because of the fragmentation of the 20% capacity left in the system. The results are consistent with the results in paper (Godfrey, 2006).

Figure 10(b) shows that the load movement factor decreases with the increase of impulse spread, and the decrease of SU. In low impulse spread, a large amount of load assigning to a small region generates a large number of overloaded nodes, so the LAR load balancing algorithm cannot handle them quickly. This situation becomes worse when SU increases to 0.8, due to little

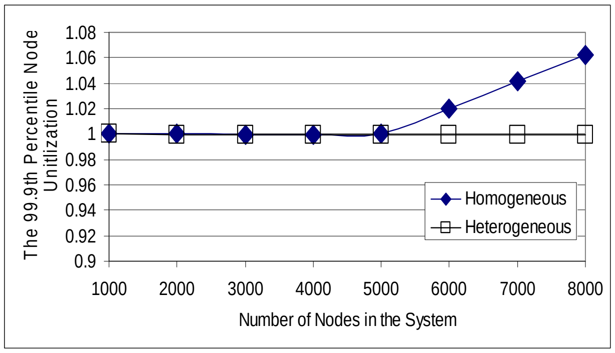


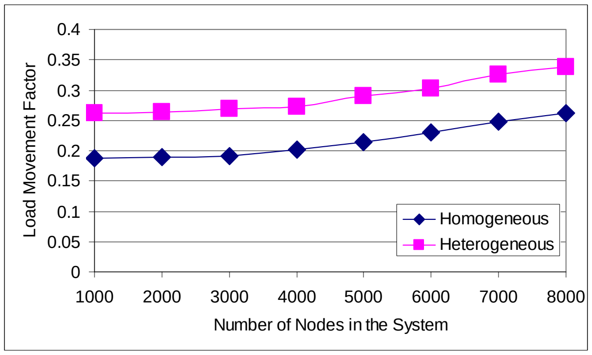
(a) The 99.9th percentile node utilization (b) Load movement factor

Figure 10. Impact of non-uniform item arrival patterns.

available capacity left. Therefore, the 99.9th NU and the load movement factor are high in highly loaded system and low impulse interval. In summary, the LAR algorithm can solve non-uniform item arrival generally. It can deal with sudden increase of 10% load in 10% ID space in a highly loaded system with SU equals to 0.8, achieving the 99.9th NU close to 1.

Impact of Node Number and Capacity Heterogeneity in Churn. Consistent hashing function adopted in DHT leads to a bound of O(log n) imbalance of keys between the nodes, where n is the number of nodes in the system. Node heterogeneity in capacity makes the load balancing problem even more severe. In this section, we study the effects of the number of nodes and heterogeneous capacity distribution in the system on load balancing. We varied the number of nodes from 1000 to 8000 at a step size of 1000, and tested NU and load movement factor when node capacities were heterogeneous and homogeneous. Homogeneous node capacities are equal capacities set as 50000, and heterogeneous node capacities are determined by the default Pareto node capacity distribution.





(a) The 99.9th percentile node utilization (b) Load movement factor

Figure 11. Impact of the number of nodes in the system.

Figure 11(a) shows that in the heterogeneous case, the 99.9th NUs are all around 1. It means that the LAR can maintain nodes to be light in different network scales when node capacities are heterogeneous. In the homogeneous case, the 99.9th NU maintains around 1 when node number is no more than 5000, but it grows linearly as node number increases when nodes are more than 5000. It is somewhat surprisingly that LAR can achieve better load balance in large scale network when node capacities are heterogeneous than when they are homogeneous. Intuitively, this is because that in the heterogeneous case, very high load items can be accommodated by large capacity nodes, but there's no node with capacity large enough to handle them in the homogeneous case. The results are consistent with those in (Godfrey, 2006).

Figure 11(b) shows that in both cases, the load movement factors increase as the number of nodes grows. Larger system scale generates higher key imbalance, such that more load needs to be transferred for load balance. The figure also shows that the factor of the homogeneous case is pronounced less than that in the heterogeneous case. This is due to the heterogeneous capacity distribution, in which some nodes have very small capacities but are assigned much higher load, which is needed to move out for load balance. The results show that node heterogeneity helps, not hurts, the scalability of LAR algorithm. LAR algorithm can achieve good load balance even in large scale network by arranging load transfer timely.

3.2.6. Summary

This section presents LAR load balancing algorithms to deal with both of the proximity and dynamism of DHTs simultaneously. The algorithms distribute application load among the nodes by "moving items" according to their capacities and proximity information in topology-aware DHTs. The LAR algorithms introduce a factor of randomness in the probing process in a range of proximity to deal with DHT churn. The efficiency of the randomized load balancing is further improved by d-way probing.

Simulation results show the superiority of a locality-aware 2-way randomized load balancing in DHTs with and without churn. The algorithm saves bandwidth in comparison with randomized load balancing because of its locality-aware feature. Due to the randomness factor in node probing, it can achieve load balance for SU as high as 90% in dynamic situations by moving load up to 20% of the system load, and up to 40% of the underlying DHT load moved caused by node joins and departures. The LAR algorithm is further evaluated with respect to a number of performance factors including load balancing frequency, arrival/departure rate of items and nodes, skewed item ID distribution, and node number and capacity heterogeneity. Simulation results show that LAR algorithm can effectively achieve load balance by moving a small amount of load even in skewed distribution of items.

4. Future Trends

Though a lot of research has been conducted in the field of load balancing in parallel and distributed systems, load balancing methods are still in their incubation phase when it comes to P2P overlay networks. In this section, we discuss the future and emerging trends, and present a number of open issues in the domain of load balancing in P2P overlay networks.

P2P overlay networks are characterized by heterogeneity, dynamism and proximity. With heterogeneity consideration, a load balancing method should allocate load among nodes based on the actual file load rather than the number of files. A dynamism-resilient load balancing method should not generate high overhead in load balancing when nodes join, leave or fail continuously and rapidly. A proximity-aware load balancing method moves load between physically close nodes so as to reduce the overhead of load balancing. However, few of the current load balancing methods take into account these three factors to improve the efficiency and effectiveness of load balancing. Virtual server methods and ID assignment and reassignment methods only aim to distribute the number of files among nodes in balance, therefore they are unable to consider file heterogeneity. In addition, these methods lead to high overhead due to neighbor maintenance and varied ID intervals owned by nodes in churn. These two categories of methods can be complementary to the load transfer methods that have potential to deal with the three features of P2P overlay networks. Thus, combing the three types of load balancing strategies to overcome each other's drawbacks and take advantage of the benefits of each method will be a promising future direction.

The LAR algorithms were built on Cycloid structured DHT. It is important that the LAR algorithms are applicable to other DHT networks as well. It must be complemented by node clustering to cluster DHT nodes together according to their physical locations to facilitate LAR's probing in a range of proximity. The work in (Shen, 2006) presents a way of clustering physically close nodes in a general DHT network, which can be applied to LAR's generalization to other DHT networks.

Currently, most heterogeneity-unaware load balancing methods measure load by the number of files stored in a node, and heterogeneity-aware load balancing methods only consider file size when determing a node's load. In addition to the storage required, the load incurred by a file also includes bandwidth consumption caused by file queries. Frequently-queried files generate high load, while infrequently-queried files lead to low load. Since files stored in the system often have different popularities, and the access patterns to the same file may vary with time, a file's load is changing dynamically. However, most load balancing methods are not able to cope with load variance caused by non-uniform and time-varying file popularity. Thus, an accurate method to measure a file's load that considers all factors affecting load is required. On the other hand, node capacity heterogeneity should also be identified. As far as the author knows, all current load balancing methods assume that there is one bottleneck resource， though there are various resources including CPU, memory, storage and bandwidth. For highly effective load balancing, the various loads such as bandwidth and storage should be differentiated, and various node resources should be differentiated as well. Rather than mapping generalized node capacity and load, corresponding load and node resource should be mapped in load balancing. These improvements will significantly enhance the accuracy and effectiveness of a load balancing method.

Most load balancing algorithms only balances key distribution among nodes. In file sharing P2P systems, a main function of nodes is to handle key location query. Query load balancing is a critical part of P2P load balancing. That is, the number of queries that nodes receive, handle and forward corresponds to their different capacities. A highly effective load balancing method will distribute both key load and query load in balance.

5. Conclusion

A load balancing method is indispensable to a high performance P2P overlay network. It helps to avoid overloading nodes and take full advantage of node resources in the system. This chapter has provided a detailed introduction of load balancing in P2P overlay networks, and has examined all aspects of load balancing methods including their goals, properties, strategies and classification. A comprehensive review of research works focusing on load balancing in DHT networks has been presented, along with an in depth discussion of their pros and cons. Furthermore, a load balancing algorithm that overcomes the drawbacks of previous methods has been presented in detail. Finally, the future and emerging trends and open issues in load balancing in P2P overlay networks have been discussed.

REFERENCES

Y. Azar, A. Broder, & et al. (1994). Balanced allocations. *In Proc. of STOC* (pp593–602).

M. Adler, E. Halperin, R. M. Karp, & V. Vazirani. (June 2003). A stochastic process on the hypercube with applications to peer-to-peer networks. *In Proc. of STOC.*

M. Bienkowski, M. Korzeniowski, & F. M. auf der Heide. (2005). Dynamic load balancing in distributed hash tables. In *Proc. of IPTPS.*

P. Brighten Godfrey, & I. Stoica. (2005). Heterogeneity and load balance in distributed hash tables. *In Proc. of IEEE INFOCOM.*

J. Byers, J. Considine, & M. Mitzenmacher. (Feb. 2003). Simple Load Balancing for Distributed Hash Tables. *In Proc. of IPTPS.*

M. Castro, P. Druschel, Y. C. Hu, & A. Rowstron. (2002). Topology-aware routing in structured peer-to-peer overlay networks. *In Future Directions in Distributed Computing.*

E. Zegura, K. Calvert, & et al. (1996). How to model an Internetwork. *In Proc. of INFOCOM.*

Fasttrack product description. (2001). http://www.fasttrack.nu/index.html.

B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, & I. Stoica. (2006). Load balancing in dynamic structured P2P systems. *Performance Evaluation*, 63(3).

K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, & I. Stoica. (2003). The impact of DHT routing geometry on resilience and proximity. *In Proc. of ACM SIGCOMM.*

S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, & L. Zhang. (2000). On the placement of Internet instrumentation. *In Proc. of INFOCOM.*

F. Kaashoek, & D. R. Karger. Koorde. (2003). A Simple Degree-optimal Hash Table. *In Proceedings IPTPS.*

D. Karger, E. Lehman, T. Leighton, M. Levine, & et al. (1997). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. *In Proc. of STOC* (pp 654–663).

D. R. Karger, & M. Ruhl. (2004). Simple efficient load balancing algorithms for Peer-to-Peer systems. In *Proc. of IPTPS.*

G. Manku. (2004). Balanced binary trees for ID management and load balance in distributed hash tables. *In Proc. of PODC.*

P. Maymounkov, & D. Mazires. Kademlia. (2002). A peer-to-peer information system based on the xor metric. *The 1st Interational Workshop on Peer-to-Peer Systems (IPTPS).*

M. Mitzenmacher. (1997). On the analysis of randomized load balancing schemes. *In Proc. of SPAA.*

R. Motwani, & P. Raghavan. (1995). *Randomized Algorithms.* Cambridge, University Press, New York, NY.

A. Mondal, K. Goda, & M. Kitsuregawa. (2003). Effective load-balancing of peer-to-peer systems. *In Proc. of IEICE DEWS DBSJ Annual Conference.*

M. Naor, & U. Wieder. (June 2003). Novel Architectures for P2P applications: the continuous-discrete approach. *In Proc. SPAA.*

A. Rao, & K. Lakshminarayanan et al. (2003). Load balancing in structured P2P systems. In *Proc. of IPTPS.*

S. Ratnasamy, M. Handley, R. Karp,, & S. Shenker. (2002). Topologically aware overlay construction and server selection. *In Proc. of INFOCOM.*

S. Ratnasamy, P. Francis, M. Handley, R. Karp, & S. Shenker. (2001). A scalable content-addressable network. *In Proceedings of ACM SIGCOMM*. (pp 329–350).

A. Rowstron, & P. Druschel. Pastry. (2001). Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *In Proc. of the 18th IFIP/ACM Int'l Conf. on Distributed Systems Platforms (Middleware).*

S. Saroiu, & et al. (2002). A Measurement Study of Peer-to- Peer File Sharing Systems. *In Proc. of MMCN.*

H. Shen, C. Xu, & G. Chen. (2006). Cycloid: A scalable constant-degree P2P overlay network. *Performance Evaluation*, 63(3) (pp195–216).

H. Shen, & C. Xu. (April 2006). Hash-based proximity clustering for load balancing in heterogeneous DHT networks. *In Proc. of IPDPS.*

H. Shen, & C.-Z. Xu. (2007). Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured Peer-to-Peer Networks. *IEEE Transactions on Parallel and Distributed Systems (TPDS).* 18(6) (pp849-862).

I. Stoica, R. Morris, & et al. (2003). Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking.*

M. Waldvogel, & R. Rinaldi. (2002). Efficient topology-aware overlay network. *In Proc. of HotNets-I.*

Xu, C. (2005). *Scalable and Secure Internet Services and Architecture.* Chapman, & Hall/CRC Press.

Z. Xu, C. Tang, & Z. Zhang. (2003). Building topology-aware overlays using global soft-state. *In Proc. of ICDCS.*

Z. Xu, M. Mahalingam, & M. Karlsson. (2003). Turning heterogeneity into an advantage in overlay routing. In *Proc. of INFOCOM*.

B. Yang, & H. Garcia-Molina. (2003). Designing a super-peer network. *In Proc. of ICDE..*

B.Y. Zhao, J. Kubiatowicz, & A.D. Oseph. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Technical Report UCB/CSD-01-1141*. University of California at Berkeley.

Y. Zhu, & Y. Hu. (2005). Efficient, proximity-aware load balancing for DHT-based P2P systems. *In Proc. of IEEE TPDS*, 16(4).

S. Fu, C. Z. Xu & H. Shen. (April 2008). Random Choices for Churn Resilient Load Balancing in Peer-to-Peer Networks. *Proc. of IEEE International Parallel and Distributed Processing Symposium.*

Key Terms and Definitions

***Peer**:* A peer (or node) is an abstract notion of participating entities. It can be a computer process, a computer, an electronic device, or a group of them.

***Peer-to-peer network**:* A peer-to-peer network is a logical network on top of physical networks in which peers are organized without any centralized coordination.

***Structured peer-to-peer network/Distributed hash table**:* A peer-to-peer network that maps keys to the nodes based on a consistent hashing function.

Heterogeneity: The instinct properties of participating peers, including computing ability, differ a lot and deserve serious consideration for the construction of a real efficient wide-deployed application.

***Proximity**:* Mismatch between logical proximity abstraction derived from DHTs and physical proximity information in reality, which is a big obstacle for the deployment and performance optimization issues for P2P applications.

***Dynamism/Churn**:* A great number of nodes join, leave and fail continually and rapidly, leading to unpredicted network size.

***Load balancing method**:* A method that controls the load in each node no more than the node's capacity.