Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured Peer-to-Peer Networks

Haiying Shen
Dept. of Computer Science and Engineering
University of Arkansas, Fayetteville, AR 72701
hshen@uark.edu

Cheng-Zhong Xu
Dept. of Electrical & Computer Engineering
Wayne State University, Detroit, MI 48202
czxu@wayne.edu

***Abstract*— Structured peer-to-peer 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. Data redistribution algorithms based on randomized matching of heavily loaded nodes with light ones can deal with the dynamics of DHTs. But they are unable to consider proximity of the nodes simultaneously. There are other methods that rely on auxiliary networks to facilitate locality-aware load redistribution. Due to the cost for network construction and maintenance, the locality-aware algorithms can hardly work for DHTs with churn. This paper presents a locality-aware randomized load balancing algorithm to deal with both of the proximity and network churn at the same time. We introduce a factor of randomness in the probing of lightly loaded nodes in a range of proximity. We further improve the efficiency by allowing the probing of multiple candidates (d-way) at a time. Simulation results show the superiority of the locality-aware 2-way randomized algorithm, in comparison with other random or locality-aware algorithms. In DHTs with churn, it performs no worse than the best churn resilient algorithm. It takes advantage of node capacity heterogeneity and achieves good load balance effectively even in a skewed distribution of items.**

***Index Terms*— Cycloid, Distributed Hash Table, Peer-to-Peer, Load Balancing, Heterogeneity, Proximity.**

I. Introduction

Over the past years, the immense popularity of peer-to-peer (P2P) resource sharing services has produced a significant stimulus to content-delivery overlay network research [20]. An important class of the overlay networks is distributed hash tables (DHTs) that map keys to the nodes of a network based on a consistent hashing function; see [17] and references therein for representatives of the DHTs. In a DHT, 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 whose IDs are located in its space range. However, consistent hashing produces a bound of $O(\log n)$ imbalance of keys between nodes, where n is the number of nodes in the system [9]. The objective of load balancing is to prevent nodes from being overloaded by distributing application load among the nodes in proportion to their capacities. Effective load balancing algorithm 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. 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. 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 [18], [13], [6], [25], [10]. However, few of them 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. Rao *et al.* [13] proposed three randomized load balancing algorithms, based on a concept of "virtual servers", and demonstrated their effectiveness in Chord. Godfrey *et al.* [6] extended the algorithms for DHTs with churn. The algorithms treat all nodes equally in random probing, without consideration of node proximity information in load balancing. Zhu and Hu presented a proximity-aware algorithm to take into account the node proximity information in load balancing [25]. The algorithm is based on an additional network constructed on top of DHTs. Although the network is self-organized, the algorithm is hardly applicable to DHTs with churn.

In this paper, we present novel locality-aware randomized (LAR) load balancing algorithms to deal with both the proximity and dynamics of DHTs. 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. We 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. We further improve the efficiency by allowing the probing of multiple candidates at a time. We refer to such probing process as d-way probing, $d \geq 1$. The algorithms are implemented in Cycloid [17], based on a concept of "moving item" [10] for retaining DHT network efficiency and scalability. We evaluated the performance of the LAR load balancing algorithms 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.

The rest of this paper is structured as follows. Section II presents a concise review of representative load balancing approaches for DHT networks. Section III briefly introduces the architecture of Cycloid related to load balancing. Section IV details a load balancing framework on Cycloid. Section V presents the LAR load balancing algorithms. Sections VI and VII show the performance of the approaches in DHTs with and without churn, respectively. Section VIII concludes this paper with remarks on future work.

II. Related Work

Load balancing is an inherent problem in any DHTs based on consistent hashing functions. Karger *et al.* proved that the consistent hashing function in chord [18] leads to a bound of $O(\log n)$ imbalance of keys between the nodes. Stoica *et al.* proposed an abstraction of "virtual servers" for Chord load balancing. This abstraction simplifies the treatment of load balancing problem at the cost of higher space overhead and lookup efficiency compromise. The original concept of "virtual servers" ignores the file size and node heterogeneity. Later on, Rao *et al.* [13] 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 load from heavy nodes to light nodes so that each node's load does not exceed its capacity. The algorithms are different primarily in the amount of information used to decide rearrangement. In the one-to-one algorithm, each light server randomly probes nodes for a match with a heavy one. In the many-to-many algorithm, each heavy server sends its excess virtual nodes to a global pool, which executes 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 heavy server randomly chooses a directory which contains information about a number of light severs. Most recently, Godfrey *et al.* [6] extended this work for dynamic DHT networks. In their approach, if a node's capacity utilization exceeds a predetermined threshold, its excess virtual servers will be moved to a light one immediately without waiting for next periodic balancing. Bienkowski *et al.* [2] proposed a node leave and re-join strategy to balance the key ID intervals across the nodes. In the algorithm, lightly loaded nodes leave the system and rejoin to share the load of heavy ones. All of these algorithms assume the objective of minimizing the amount of moved load. They 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 work to utilize the proximity information to guide load balancing is due to Zhu and Hu [25]. 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 disseminate final information to all the virtual nodes. Using this information, each real server can determine whether it is heavily loaded or not. Light and heavy 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. The KT structure helps to use proximity information to move load between physically close heavy and light 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 solved before its recovery. 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.

Most recently, Karger and Ruhl [10] proved that the "virtual servers" method could not be guaranteed to handle item distributions where an 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 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 heavy node to the light 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 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 $O(1/n)$ [10].

This paper presents LAR algorithms that take into account proximity information in load balancing and deal with network dynamism meanwhile. A first implementation of item movement based the algorithms is also reported, although the algorithms are also suitable for "virtual servers". The implementation approach bears similarity to the one in [12] by viewing a P2P system as a cluster of clusters. The latter was designed for unstructured P2P systems.

III. Cycloid: A Constant-Degree DHT

Cycloid [17] is a lookup efficient constant-degree DHT that we recently proposed. In a Cycloid system with $n = d \cdot 2^d$ 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. For more information about Cycloid, please refer to [17].

A. 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 $a_{d-1}a_{d-2}......a_0$ is a cubical index. The cyclic index is an integer, ranging from 0 to $d-1$ and the cubical index is a

TABLE I
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)

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 I 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 $(k, a_{d-1}a_{d-2} \ldots a_k \ldots a_0)$, $k \neq 0$, has one cubical neighbor $(k-1, a_{d-1}a_{d-2} \ldots \overline{a}_k xx...x)$ where x denotes an arbitrary bit value, and two cyclic neighbors $(k-1, b_{d-1}b_{d-2} \ldots b_0)$ and $(k-1, c_{d-1}c_{d-2} \ldots c_0)$. 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, $b_{d-1} \ldots b_1 b_0$)
= $\min\{\forall$(k-1, $y_{d-1} \ldots y_1 y_0)|y_{d-1} \ldots y_0 \geq a_{d-1} \ldots a_1 a_0\}$,
(k-1, $c_{d-1} \ldots c_1 c_0$)
= $\max\{\forall$(k-1, $y_{d-1} \ldots y_1 y_0)|y_{d-1} \ldots y_0 \leq a_{d-1} \ldots a_1 a_0\}$.
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.

B. 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.

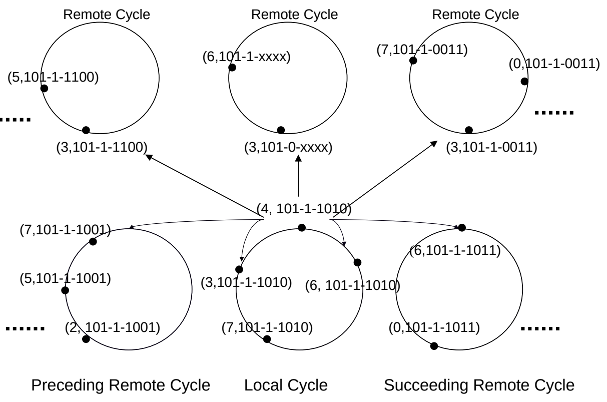


Fig. 1. Cycloid node routing links state.

IV. 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 [4], [23]. 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. We designate a node as supernode if its capacity is higher than a pre-defined threshold. To ensure every primary node meets the capacity requirement of supernodes, we modify the Cycloid rules for node join and leave slightly. 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 super node 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 super node 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.* $L_i \leq C_i$) 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, labelled as D$_k$, $k = 1, 2, \ldots$. 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 $\{D_1, D_2, \ldots, D_{m'}\}$, $1 \leq m' \leq m_i$. Their corresponding loads are $\{L_{i,1}, ..., L_{i,m'}\}$. The set of excess items is determined in such a way that

$$\text{minimizes } \sum_{k=1}^{m'} L_{i,k}, \tag{1}$$

$$\text{subject to } (L_i - \sum_{k=1}^{m'} L_{i,k}) \leq C_i. \tag{2}$$

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 II. 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: $< L_{i,1}, D_{i,1}, A_i >, < L_{i,k}, D_{i,k}, A_i >, \ldots, < L_{i,m'}, D_{i,m'}, A_i >$, in which A_i denotes the IP address of node i. Load information of light node j is represented in the form of $< \delta L_j, A_j >$. An SSL is sorted in a descending order of $L_{i,k}$; $minL_{i,k}$ represents the item with the minimum load in the primary node's starving list. A DSL is sorted in an ascending order of δL_j; $\max \delta L_j$ represents the maximum δL_j in the primary node's donating list. Load rearrangement is executed between a pair of DSL and SSL, as shown in Algorithm 1.

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 δL_j 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 nodes" 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.

TABLE II
DONATING AND STARVING SORTED LISTS.

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

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 $\delta L_j - L_{i,j} > 0$ 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, 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 don'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 [13]. This scheme can be extended to perform load rearrangement between one SSL and multiple DSLs for improvement.

V. 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.

A. 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 between 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.* [1] 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 modelled 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.* [11] 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 had 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 [5], 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.

B. 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 [3].

We integrate proximity-neighbor selection and topologically-aware overlay construction techniques in [22], [3] and [19] 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 VI.

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

TABLE III
SIMULATION SETTINGS AND ALGORITHM PARAMETERS.

Environment Parameter	**Default value**
Object arrival location	Uniform over ID space
Number of nodes	4096
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

node and its sequential nodes are usually smaller than distances between the node and randomly chosen nodes in the entire ID space.

VI. PERFORMANCE EVALUATION

We designed and implemented a simulator in Java for evaluation of the load balancing algorithms on topology-aware Cycloid. Table III 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 be evaluated in Section VII.

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 n 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 message size times physical path length of the message travelled. 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.

A. Topology-aware Cycloid Construction

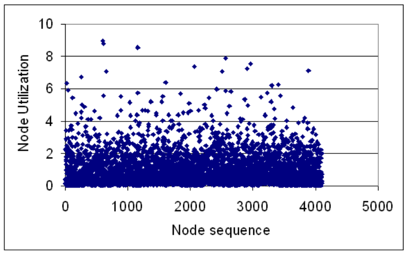
GT-ITM (transit-stub and tiers) [24] is a network topology generator, widely used for the construction of topology-aware overlay networks [14], [22], [21], [7]. We used GT-ITM to generate transit-stub topologies for Cycloid, and get physical hop distance for each pair of Cycloid nodes. Recall that we 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.

We use landmark clustering and Hilbert number [22] 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 cubic 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 [8] would only serve to improve our results.

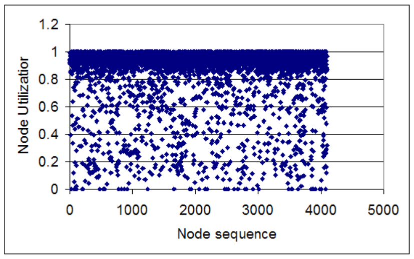
Our experiments are built on two transit-stub topologies: "ts5k-large" and "ts5k-small" with approximately 5,000 nodes each. In the topologies, nodes are organized into logical domains. We classify the domains into two types: transit domains and stub domains. Nodes in a stub domain are typically an endpoint 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.

B. Effectiveness of LAR Algorithms

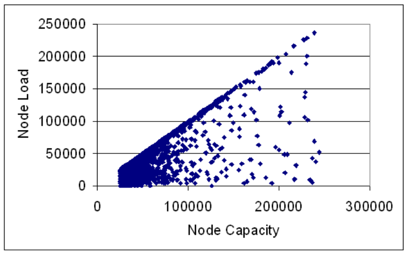
In this section, we will show the effectiveness of LAR load balancing algorithm. First, we present the impact of LAR algorithm on the alignment of the skews in load distribution and node capacity when the system is fully loaded. Figure 2(a)



(a) Before load balancing



(b) After load balancing



(c) Node load after balancing

Fig. 2. Effect of load balancing.

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 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 servers” 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.* [13] 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, we will only show results of them in “ts5k-large”.

C. Comparison with Other Algorithms

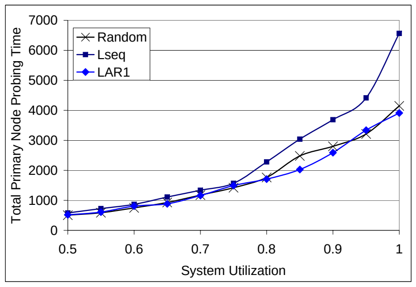
Figure 3(a) shows the probing process in Lseq takes much more time than R_1 and LAR_1. This implies that random algorithm is better than sequential algorithm in probing efficiency. Figure 3(b) shows that the rearrangement number of the three algorithms are almost the same. This implies that they need 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 [11] 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.

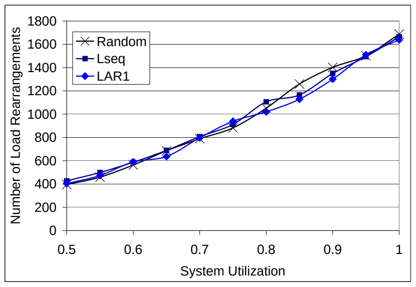
Figure 3(e) and (f) show the performance of the algorithms in “ts5k-small”. These results also confirm that LAR_1 achieve better locality-aware performance than R_1, although the improvement is not so significant as in “ts5k-large”. It is because that in “ts5k-small” topology, nodes are scattered in 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 K-nary Tree (KTree) algorithm [25] and churn resilient algorithm (CRA) [6] for comparison. From the figures, we can see that LAR performs as well as KTree, and outperform 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.

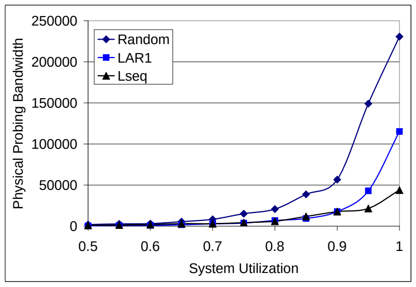
In summary, the results in Figure 3 suggest that the randomized algorithm is more efficient than the sequential algorithm in the probing process. The locality-aware approaches can effectively assign and transfer loads between neighboring nodes first, thereby reduce network traffic and improve load balancing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



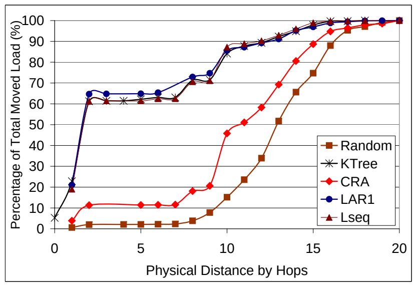
(a) Total primary node probing in "ts5k-large"



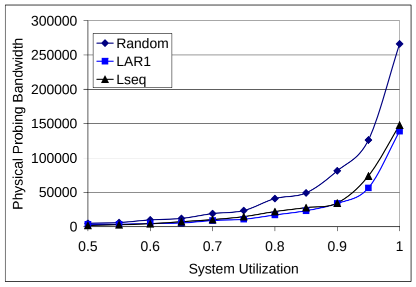
(b) Total number of load in "ts5k-large"



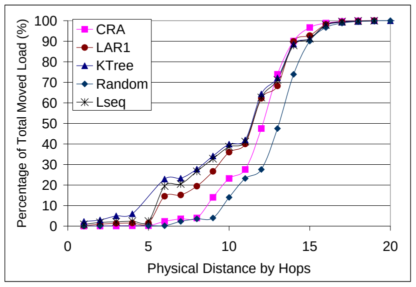
(c) Total bandwidth of time in "ts5k-large"



(d) CDF of moved load distribution in "ts5k-large"



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



(f) CDF of moved load distribution in "ts5k-small"

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

D. 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.

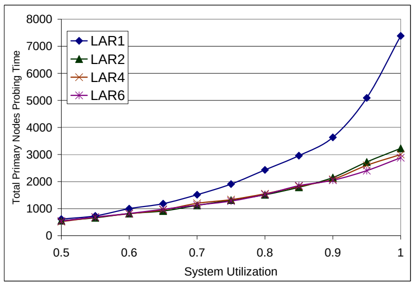
Figures 5(a) and (b) show that the probing efficiency of LAR$_d$ (d>2) is almost the same as LAR$_2$, though they need to probe more nodes than LAR$_2$. Our results are consistent with our expectations in Section V-A that a two-way probing method leads to an exponential improvement over one-way probing, 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 LAR$_1$ 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.

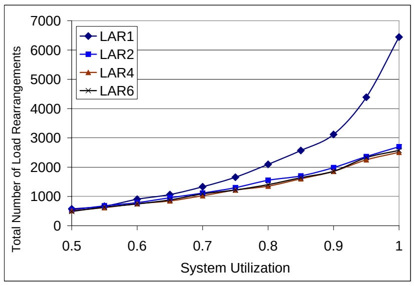
VII. Load Balancing in DHTs with Churn

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 with CRA, which was designed for DHTs with churn. The performance factors we considered include load balancing frequency, item arrival/departure rate, nonuniform item arrival pattern, and network scale and node capacity heterogeneity. We adopted the same metrics as in [6]:

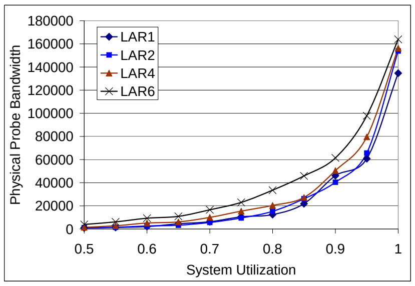
1) *The 99.9th percentile node utilization (99.9th NU)*. We



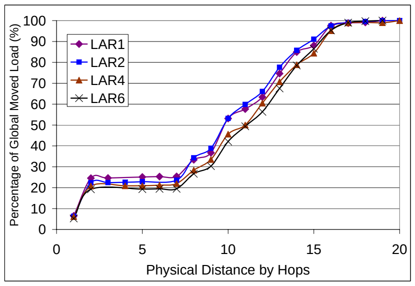
(a) Total primary node probing in “ts5k-large”



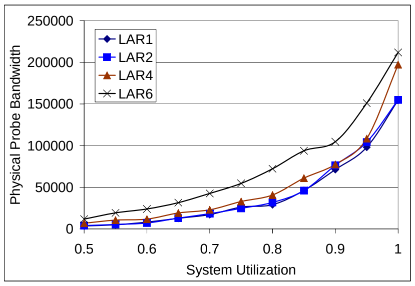
(b) Total number of load in “ts5k-large”



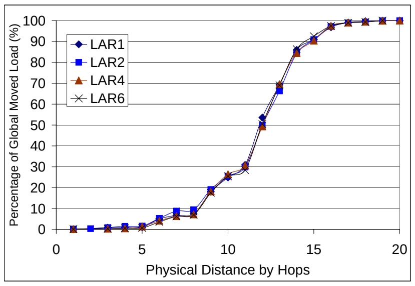
(c) Total bandwidth of probings in “ts5k-large”



(d) CDF of moved load distribution in “ts5k-large”

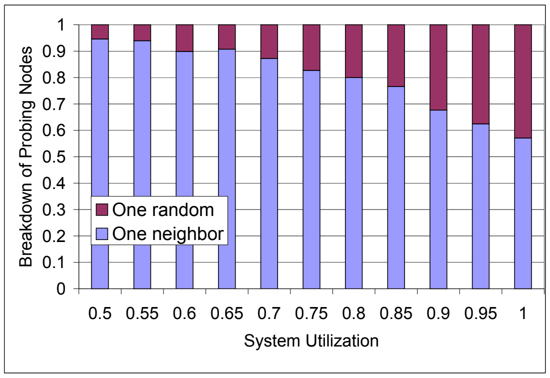


(e) Total bandwidth of probings in “ts5k-small”

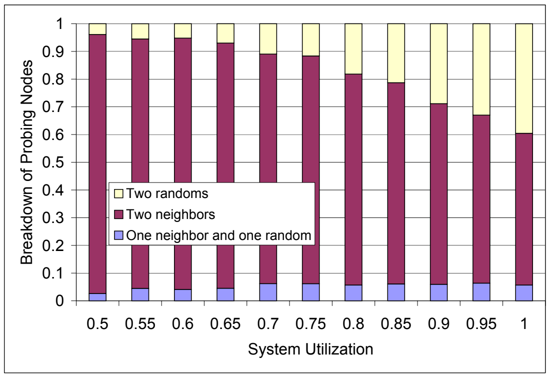


(f) CDF of moved load distribution in “ts5k-small”

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



(a) LAR_1



(b) LAR_2

Fig. 4. Breakdown of probed nodes.

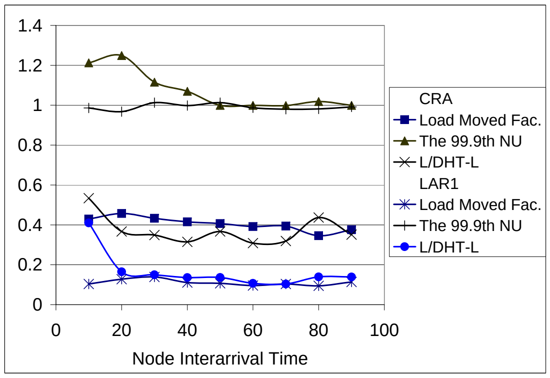


Fig. 6. Effect of load balancing with churn.

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 modelled 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.

A. Performance Comparison With CRA

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 LAR_1 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.

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 LAR algorithm.

B. Impact of Load Balancing Frequency

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. 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.

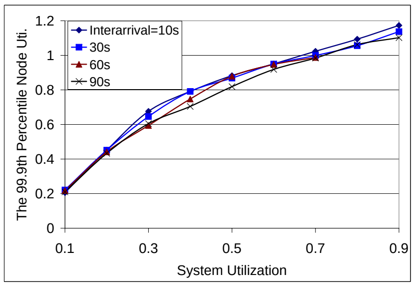
Figure 8(a) shows that all the 99.9th NUs are less than 1, and when the actual load of a system consists more than 60% of its target load, the 99.9 NU quickly converges to 1. It implies that the LAR algorithm is effective in keeping every node light, and it can quickly transfer excess load of heavy nodes 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 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.

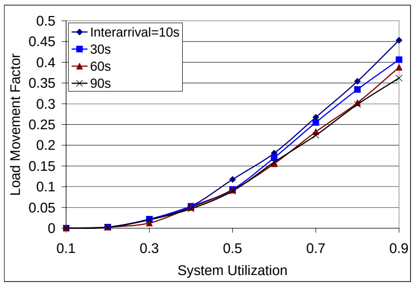
C. Impact of Item Arrival/Departure Rate

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 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.

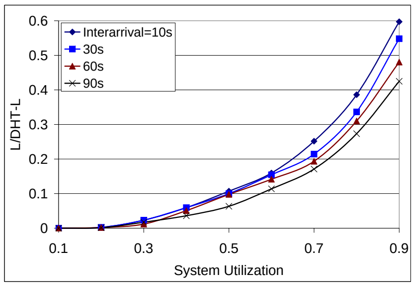
This is confirmed by the observation in Figure 9(b) that the load moved is higher in heavily loaded system than that in lightly loaded system, and movement factor drops faster



(a) The 99.9th percentage NU

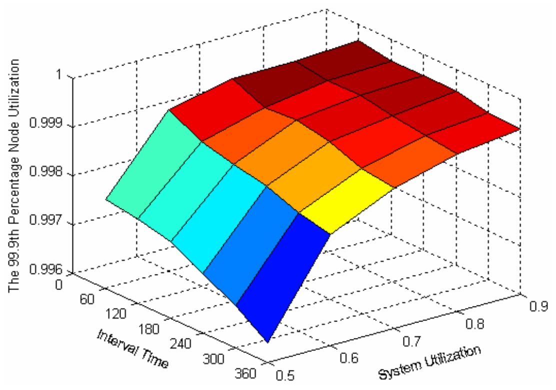


(b) Load movement factor

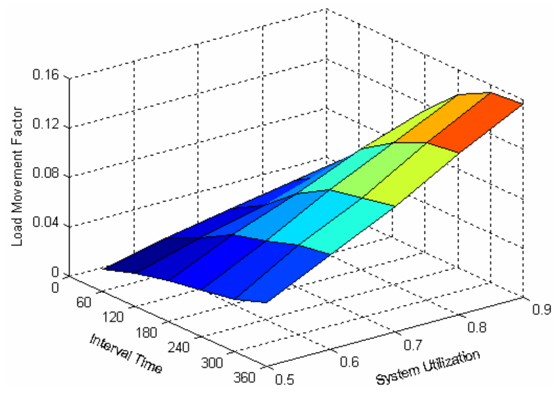


(c) L/DHT-L

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

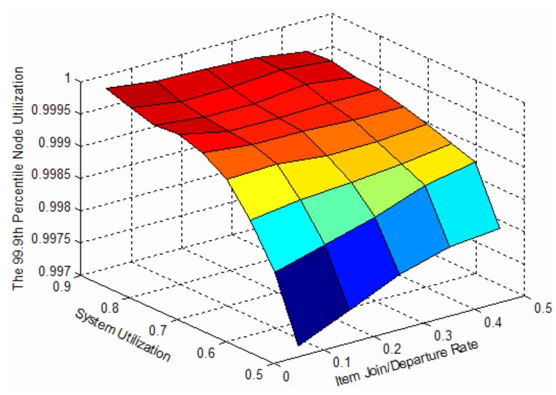


(a) The 99.9th percentage node utilization

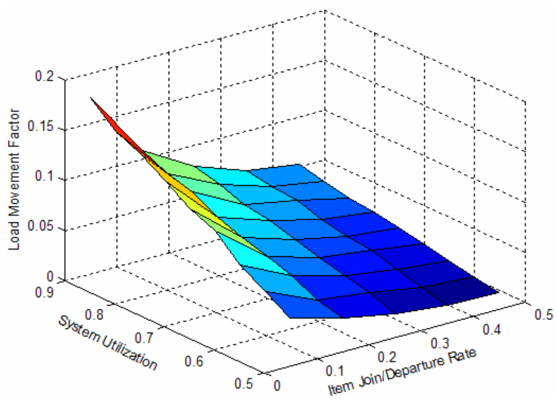


(b) Load movement factor

Fig. 8. Impact of load balancing frequency.



(a) The 99.9th percentage node utilization



(b) Load movement factor

Fig. 9. Impact of item arrival/departure rate.

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.

D. Impact of Nonuniform Item Arrivals

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 a ID space interval. We set their total load as 10% of the total system load, and varied the spread of interval from 10% 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 solved 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 [6]. 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 balancer cannot handle them quickly. This situation becomes worse when SU increases to 0.8, due to little 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 nonuniform 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.

E. Impact of Node Number and Capacity Heterogeneity

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 make 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.

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 [6].

Figure 11(b) shows that in both cases, the load movement factors increase as node number 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 heterogenous 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.

VIII. Conclusions

This paper presents LAR load balancing algorithms to deal with both of the proximity and dynamics 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. We introduce a factor of randomness in the probing process in a range of proximity to deal with DHT churn. We further improve the randomized load balancing efficiency 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. We further evaluate the LAR algorithm 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.

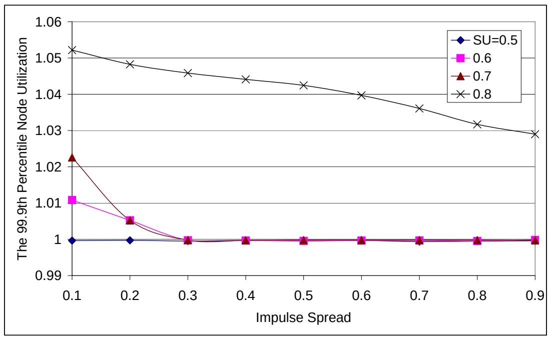
Although the LAR algorithm was tested on Cycloid-structured DHTs, it is applicable to other DHT networks, as well. It must be complemented by node clustering to cluster DHT nodes together according to their physical locations so as to facilitate LAR's probing in a range of proximity. Section 6.1 presented a way of clustering in Cycloid. Readers are referred to [16] for its generalization to other DHT networks.

We also note that the load balancing algorithms work for key distribution load balancing. 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 is based on their different capacities accordingly. We will explore methods for this.

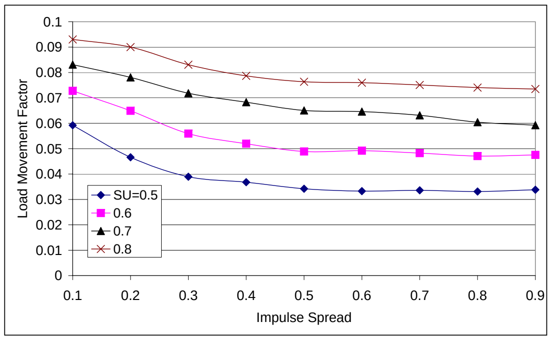
Acknowledgments: We are grateful to the anonymous reviewers for constructive comments. This research was supported in part by U.S. NSF grants ACI-0203592, CCF-0611750, and MCS-0624849 and an NASA grant 03-OBPR-01-0049. Preliminary results of this work were presented in [15].

References

[1] Y. Azar, A. Broder, and et al. Balanced allocations. In *Proc. of STOC*, pages 593–602, 1994.
[2] M. Bienkowski, M. Korzeniowski, and F. M. auf der Heide. Dynamic load balancing in distributed hash tables. In *Proc. of IPTPS*, 2005.

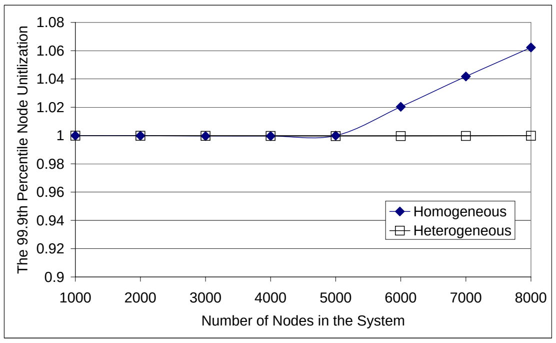


(a) The 99.9th percentage node utilization

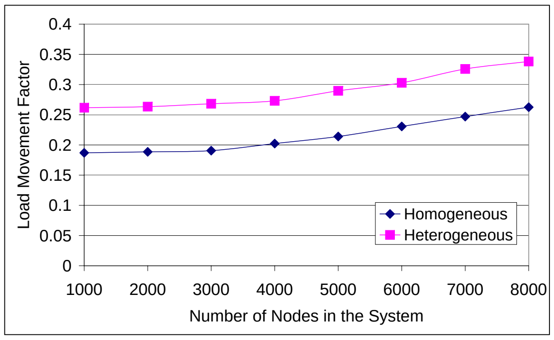


(b) Load movement factor

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



(a) The 99.9th percentage node utilization



(b) Load movement factor

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

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

[4] Fasttrack product description, 2001. http://www.fasttrack.nu/index_int.html.

[5] S. Fu, C. Xu, and H. Shen. Random choices for churn resilient load balancing in peer-to-peer networks. Technical report, ECE Department, Wayne State University, September 2006.

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

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

[8] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of Internet instrumentation. In *Proc. of INFOCOM*, 2000.

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

[10] D. R. Karger and M. Ruhl. Simple efficient load balancing algorithms for Peer-to-Peer systems. In *Proc. of IPTPS*, 2004.

[11] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proc. of SPAA*, 1997.

[12] A. Mondal, K. Goda, and M. Kitsuregawa. Effective load-balancing of peer-to-peer systems. In *Proc. of IEICE DEWS DBSJ Annual Conference*, 2003.

[13] A. Rao and K. Lakshminarayanan et al. Load balancing in structured P2P systems. In *Proc. of IPTPS*, 2003.

[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. of INFOCOM*, 2002.

[15] H. Shen and C. Xu. Locality-aware randomized load balancing algorithms for structured P2P networks. In *Proc. of ICPP*, pages 529–536, 2005.

[16] H. Shen and C. Xu. Hash-based proximity clustering for load balancing in heterogeneous DHT networks. In *Proc. of IPDPS'06*, April 2006.

[17] H. Shen, C. Xu, and G. Chen. Cycloid: A scalable constant-degree p2p overlay network. *Performance Evaluation*, 63(3):195–216, 2006. An early version appeared in Proc. of IPDPS'04.

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

[19] M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. In *Proc. of HotNets-I'02*.

[20] C. Xu. Scalable and Secure Internet Services and Architecture. Chapman & Hall/CRC Press, 2005.

[21] Z. Xu, M. Mahalingam, and M. Karlsson. Turning heterogeneity into an advantage in overlay routing. In *Proc. of INFOCOM*, 2003.

[22] Z. Xu, C. Tang, and Z. Zhang. Building topology-aware overlays using global soft-state. In *Proc. of ICDCS*, 2003.

[23] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proc. of ICDE*, 2003.

[24] E. Zegura, K. Calvert, and et al. How to model an Internetwork. In *Proc. of INFOCOM*, 1996.

[25] Y. Zhu and Y. Hu. Efficient, proximity-aware load balancing for dht-based p2p systems. *IEEE TPDS*, 16(4), 2005. An early version appeared in Proc. of IPDPS'04.