Available online at www.sciencedirect.com
SCIENCE DIRECT®
Performance Evaluation 63 (2006) 195–216
PERFORMANCE EVALUATION
An International Journal
www.elsevier.com/locate/peva

Cycloid: A constant-degree and lookup-efficient P2P overlay network

Haiying Shen[a], Cheng-Zhong Xu[a,*], Guihai Chen[b]

[a] *Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202, USA*
[b] *State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, China*

Available online 19 March 2005

Abstract

There are many structured P2P systems that use DHT technologies to map data items onto the nodes in various ways for scalable routing and location. Most of the systems require O(log n) hops per lookup request with O(log n) neighbors per node, where n is the network size. In this paper, we present a constant-degree P2P architecture, namely *Cycloid*, which emulates a cube-connected cycles (CCC) graph in the routing of lookup requests. It achieves a time complexity of O(d) per lookup request by using O(1) neighbors per node, where $n = d \times 2^d$. We compare Cycloid with other two constant-degree systems, Viceroy and Koorde in various architectural aspects via simulation. Simulation results show that Cycloid has more advantages for large scale and dynamic systems that have frequent node arrivals and departures. In particular, Cycloid delivers a higher location efficiency in the average case and exhibits a more balanced distribution of keys and query loads between the nodes.

Keywords: Cycloid; Peer-to-peer; Viceroy; Koorde; Distributed hash table; Constant-degree DHT

1. Introduction

Over the past years, the immerse popularity of peer-to-peer resource sharing services has produced a significant stimulus to content-delivery overlay network research. 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

* Corresponding author. Tel.: +1 313 577 3856; fax: +1 313 578 5853.
E-mail addresses: shy@ece.eng.wayne.edu (H. Shen); czxu@ece.eng.wayne.edu (C.-Z. Xu); gchen@nju.edu.cn (G. Chen).

0166-5316/$ – see front matter © 2005 Elsevier B.V. All Rights Reserved.
doi:10.1016/j.peva.2005.01.004

hashing function. Representatives of the DHTs include CAN [16], Chord [19], Pastry [17], Tapestry [22] and Kademlia [13]. They organize the nodes in various ways for efficient location of data items. Most of the DHTs require O(log n) hops per lookup request with O(log n) neighbors per node, where n is the network size.

The network degree determines the number of neighbors with which a node must maintain continuous contact. In order to reduce the cost for maintenance, in this paper we present a new constant-degree DHT, namely Cycloid. It achieves a lookup path length of O(d) with O(1) neighbors, where d is the network dimension and $n = d \times 2^d$. It combines Chord and Pastry and emulates a cube-connected cycles (CCC) graph in the routing of lookup requests between the nodes.

There exist other two constant-degree DHTs: Viceroy [12] and Koorde [9]. Both of them feature a time complexity of O(log n) hops per lookup request with O(1) neighbors per node. But they are different in maintenance of the connectivity between a changing set of nodes and in routing for efficient key location. Koorde embeds a de Bruijn graph on the identifier circle for forwarding lookup requests. It bears much resemblance to Chord in routing and connectivity maintenance. Viceroy emulates a butterfly network by assuming a real number ID space in [0, 1). It requires to select a butterfly level parameter of each node according to an estimate of the network size. Due to the dynamic nature of peer-to-peer systems, the level of a node may change with time. By contrast, Cycloid specifies each node by a pair of cyclic and cubic indices. It emulates a CCC graph by using a routing algorithm similar to the one in Pastry. Although the lookup complexity of all the three constant-degree DHTs are of the same order O(log n), our simulation results show that Cycloid has a much shorter path length per lookup request in the average case than Viceroy and Koorde. Cycloid distributes keys and lookup load more evenly between the participating nodes than Viceroy. Also, Cycloid is more robust as it continues to function correctly and efficiently with frequent node joins and leaves.

The rest of this paper is structured as follows. Section 2 presents a concise review of representative DHTs. In particular, Viceroy and Koorde are discussed in detail. Section 3 details the architecture of Cycloid, with an emphasis on its nodal routing table, routing algorithm, and self-organization considerations. Section 4 shows the performance of Cycloid, in comparison with Viceroy and Koorde. Finally, Section 5 concludes this paper with remarks on possible future work.

2. Related work

There are two classes of peer-to-peer content-delivery overlay networks: unstructured and structured. Unstructured networks such as Gnutella [2] and Freenet [1,5,6] do not assign responsibility for data to specific nodes. Nodes join and leave the network according to some loose rules. There are recent improved query methods, including flooding [2,7] where the query is propagated to all neighbors and random-walkers [10] where the query is forwarded to randomly chosen neighbors until the object is found.

Flooding-based search mechanism brings about heavy traffic in a large-scale system because of exponential increase in messages generated per query. Though random-walkers reduce flooding by some extent, they still create heavy overhead to the network due to the many forwarding peers involved. Furthermore, flooding and random walkers cannot guarantee data location. They cannot ensure query termination either once the data is located. In random walkers, a satisfied query cannot stop the other queries to other random neighbors for the same content. Both of the approaches cannot prevent one node from receiving

the same query multiple times, thus wasting bandwidth. Yang et al. studied related issues and designed a scalable hybrid P2P system based on the concepts of chained architecture, replication and hash [21]. Unstructured networks are simple in organization and able to ensure high rate churn. They also support keyword searches.

Structured networks have strictly controlled topologies. The data placement and lookup algorithms are precisely defined based on a distributed hash table (DHT) data structure. The node responsible for a key can always be found even if the system is in a continuous state of change. Because of their potential efficiency, robustness, scalability and deterministic data location, structured networks have been studied intensively in recent years. Representative DHTs include CAN [16], Chord [19], Pastry [17], Tapestry [22] and Kademlia [13]. Structured networks have three main disadvantages. First, DHT is less adept at supporting keyword searches because it is designed for exact-match query. Second, it incurs substantial repair operations for high churn rate. Third, hot-spots are generated for too frequently accessed files. Readers are referred to [18] for more discussions about advantages and disadvantages of structured and unstructured P2P networks.

In the following, we review and compare structured DHT representatives by focusing on their topological aspects. Space limitation prevents us from a detailed discussion of each system. Instead, we give more detailed descriptions of constant-degree Viceroy and Koorde DHTs for comparison. Hypercube-based Pastry is discussed in detail, as well because it serves as a base of our Cycloid DHT.

2.1. Hypercube-based

Plaxton et al. [14] developed perhaps the first routing algorithm that could be scalably used for P2P systems. Tapestry and Pastry use a variant of the algorithm. The approach of routing based on address prefixes, which can be viewed as a generalization of hypercube routing, is common to all theses schemes. The routing algorithm works by correcting a single digit at a time in the left-to-right order. If node number 12,345 received a lookup query with key 12,456, which matches the first two digits, then the routing algorithm forwards the query to a node which matches the first three digits (e.g., node 12,467). To do this, a node needs to have, as neighbors, nodes that match each prefix of its own identifier but differ in the next digit. For each prefix (or dimension), there are many such neighbors (e.g., node 12,467 and node 12,478 in the above case) since there is no restriction on the suffix, i.e., the rest bits right to the current bit. This is the crucial difference from the traditional hypercube connection pattern and provides the abundance in choosing cubical neighbors and thus a high fault resilience to node absence or node failure.

Besides such cubical neighbors spreading out in the key space, each node in Pastry also contains a leaf set L of neighbors which are the set of $|L|$ numerically closest nodes (half smaller, half larger) to the present node ID and a neighborhood set M which are the set of $|M|$ geographically closest nodes to the present node.

2.2. Ring-based

Chord uses a one-dimensional circular key space. The node responsible for the key is the node whose identifier most closely follows the key numerically; that node is called the key's successor. Chord maintains two sets of neighbors. Each node has a successor list of k nodes that immediately follow it in the key space and a finger list of O(log n) nodes spaced exponentially around the key space. The ith entry of

the finger list points to the node that is 2^i away from the present node in the key space, or to that node's successor if that node is not alive. So the finger list is always fully maintained without any null pointer. Routing correctness is achieved with such two lists. A lookup(key) is, except at the last step, forwarded to the node closest to, but not past, the key. The path length is O($\log n$) since every lookup halves the remaining distance to the home.

2.3. Mesh-based

CAN chooses its keys from a d-dimensional toroidal space. Each node is associated with a region of this key space, and its neighbors are the nodes that own the contiguous regions. Routing consists of a sequence of redirections, each forwarding a lookup to a neighbor that is closer to the key. CAN has a different performance profile than the other algorithms; nodes have O(d) neighbors and path lengths are O($dn^{1/d}$) hops. Note that when $d = \log n$, CAN has O($\log n$) neighbors and O($\log n$) path length like the other algorithms. This actually gives another way to deploy the hypercube as an overlay network.

Recently, Xu et al. [20] studied the fundamental tradeoff between the routing table size and the network diameter and proved that the Tapestry, Pastry, Chord and CAN schemes were indeed asymptotically optimal as uniform algorithms. They found a routing table size threshold ($\log n$) that separates the tradeoff region dominated by congestion and the region dominated by reachability for uniform algorithms. They further proposed a graph based on a modified static butterfly that costs O$\left(\frac{\log n}{\log\log n}\right)$ and O($\log_d n$) for lookups with routing tables of size $\log n$ and d, respectively.

2.4. Constant-degree DHTs

Viceroy [12] maintains a connection graph with a constant-degree logarithmic diameter, approximating a butterfly network. Each Viceroy node in butterfly level l has seven links to its neighbors, including pointers to its predecessor and successor pointers in a general ring, pointers to the next and previous nodes in the same level ring, and butterfly pointers to its left, right nodes of level $l + 1$, and up node of level $l - 1$, depending on the node location.

In Viceroy, every participating node has two associated values: its identity $\in [0, 1)$ and a butterfly level index l. The node ID is independently and uniformly generated from a range $[0, 1)$ and the level is randomly selected from a range of $[1, \log n_0]$, where n_0 is an estimate of the network size. The node ID of a node is fixed, but its level may need to be adjusted during its life time in the system.

Viceroy routing involves three steps: ascending to a level 1 node via up links, descending along the down link until a node is reached with no down links, and traversing to the destination via the level ring or ring pointers. Viceroy takes O($\log n$) hops per lookup request.

Koorde [9] combines Chord with de Bruijn graphs. Like Viceroy, it looks up a key by contacting O($\log n$) nodes with O(1) neighbors per node. As in Chord, a Koorde node and a key have identifiers that are uniformly distributed in a 2^d identifier space. A key k is stored at its successor, the first node whose ID is equal to or follows k in the identifier space. Node $2^d - 1$ is followed by node 0.

Due to the dynamic nature of the P2P systems, they often contain only a few of the possible 2^d nodes. To embed a de Bruijn graph on a sparsely populated identifier ring, each participating node maintains knowledge about its successor on the ring and its first de Bruijn node. To look up a key k, the Koorde

Table 1
A comparison of some representative P2P DHTs

Systems	Base network	Lookup complexity	Routing table size
Chord	Cycle	O(log n)	O(log n)
CAN	Mesh	O($d \times n^{1/d}$)	O(d)
eCAN	Mesh	O($d \times n^{1/d}$)	O(d)
Pastry/Tapestry	Hypercube	O(log n)	O($\|L\|$) + O($\|M\|$) + O(log n)
Viceroy	Butterfly	O(log n)	7
Koorde	de Bruijn	O(log n)	$\geq$2
Cycloid	CCC	O(d)	7

routing algorithm must find the successor of k by walking down the de Bruijn graph. Since the de Bruijn graph is usually incomplete, Koorde simulates the path taken through the complete de Bruijn graph, passing through the immediate real predecessor of each imaginary node on the de Bruijn path.

We summarize the architectural characteristics of the representative DHTs in Table 1.

3. Cycloid: a constant-degree DHT

Cycloid combines Pastry with CCC graphs. In a Cycloid system with $n = d \times 2^d$ nodes at most, each lookup takes O(d) hops with O(1) neighbors per node. Note that the Cycloid is not necessarily complete; it can have nodes less than $d \times 2^d$ with some void node places. Like Pastry, it employs consistent hashing to map keys to nodes. A node and a key have identifiers that are uniformly distributed in a $d \times 2^d$ identifier space.

3.1. CCC and key assignment

A d-dimensional CCC graph is a d-dimensional cube with replacement of each vertex by a cycle of d nodes. It contains $d \times 2^d$ nodes of degree 3 each. Each node is represented by a pair of indices $(k, a_{d-1}a_{d-2}\cdots a_0)$, where k is a cyclic index and $a_{d-1}a_{d-2}\cdots a_0$ 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. Fig. 1 shows the three-dimensional CCC.

A P2P system often contains a changing set of nodes. This dynamic nature poses a challenge for DHTs to manage a balanced distribution of keys among the participating nodes and to connect the nodes in an easy-to-maintain network so that a lookup request can be routed toward its target quickly. In a Cycloid system, each node keeps a routing table and two leaf sets with a total of seven entries to maintain its connectivity to the rest of the system. Table 2 shows a routing state table for node (4,101–1–1010) in an eight-dimensional Cycloid. Its corresponding links in both cubical and cyclic aspects are shown Fig. 2.

In general, a node $(k, a_{d-1}a_{d-2}\cdots a_k\cdots a_0)$ $(k \neq 0)$ has one cubical neighbor $(k-1, a_{d-1}a_{d-2}\cdots \bar{a}_k x x\cdots x)$ where x denotes an arbitrary bit value, and two cyclic neighbors $(k-1, b_{d-1}b_{d-2}\cdots b_0)$ and $(k-1, c_{d-1}c_{d-2}\cdots c_0)$. 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 is no larger than $k-1$. That is:

$$(k-1, b_{d-1}\cdots b_1 b_0) = \min\{\forall(k-1, y_{d-1}\cdots y_1 y_0) | y_{d-1}\cdots y_0 \geq a_{d-1}\cdots a_1 a_0\},$$

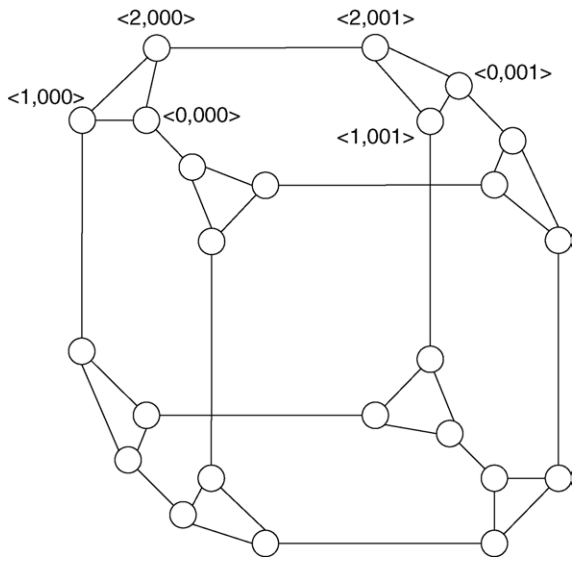

Fig. 1. A three-dimensional cube-connected cycles.

$$(k-1, c_{d-1}\cdots c_1c_0) = \max\{\forall(k-1, y_{d-1}\cdots y_1y_0) | y_{d-1}\cdots y_0 \le a_{d-1}\cdots a_1a_0\}$$

The node with a cyclic index $k = 0$ has no cubical neighbor and 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 cycle. The left inside leaf set node points to the node's predecessor and the right inside leaf set node points to the node's successor in the local cycle. The largest cyclic index node in a local cycle is called the primary node of the local cycle. All local cycles are ordered by their cubical index mod 2^d on a large cycle. The left outside leaf set node points to the primary node in the node's preceding remote cycle and the right outside leaf set node points to the primary node in the node's succeeding remote cycle in the large cycle. In other words, the nodes with the same cubical index are connected by a local cycle, and all the nodes with different cubical indices are connected by a large ring. The nodes on one cycle can reach the nodes in other cycles directly or indirectly.

Table 2
Routing table state of a Cycloid node (4,101–1–1010)

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)

x indicates an arbitrary value, 0 or 1. Inside leaf set maintains the node's predecessor and successor in the local cycle. Outside leaf set maintains the links to the preceding and the succeeding remote cycles.

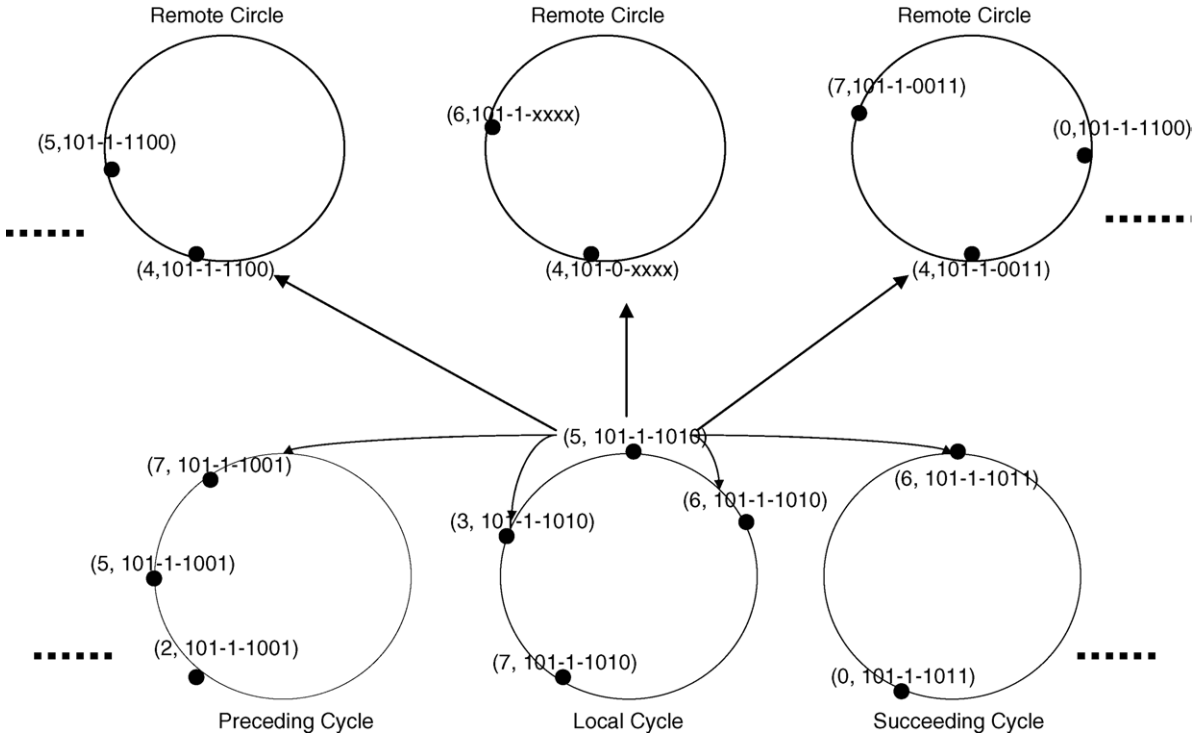

Fig. 2. Cycloid node routing links state.

The cubical links allow us to change cubical index from left to right, in the same left-to-right order as in Pastry. The cyclic links allow us to change the cyclic index. It is easy to see that the network will be the traditional cube-connected cycles if all nodes are alive. Our connection pattern is resilient in the sense that even if many nodes are absent, the remaining nodes are still capable of being connected. The routing algorithm is heavily assisted by the leaf sets. The leaf sets help improve the routing efficiency, check the termination condition of a lookup, and wrap around the key space to avoid the target overshooting. How the routing table and leaf sets are initialized and maintained is the subject of Section 3.3.

The Cycloid DHT assigns keys onto its ID space by the use of a consistent hashing function. The key assignment is similar to Pastry, except that the Cycloid associates a pair of cyclic and cubic indices with each node. 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 Cycloid peers constitutes a complete CCC graph, the consistent hashing will map key k to node k and a lookup request can be redirected in O(d) steps following the Cycloid routing algorithm. If the target node of a key's ID $(k, a_{d-1} \cdots a_1 a_0)$ is not a participant, the key is assigned to the node whose ID is first numerically closest to $a_{d-1}a_{d-2} \cdots a_0$ and then numerically closest to k. For example, (1,1101) is closer to (2,1101) than (2,1001). In the case of two nodes with the same distance to the key's ID, the key's successor will be responsible for storing this key. Cycloid key storage mechanism is almost the same as that of Pastry [17].

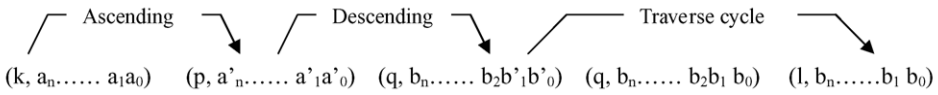

Fig. 3. Lookup algorithm in Cycloid.

3.2. Cycloid routing algorithm

Cycloid routing algorithm emulates the routing algorithm of CCC [15] from source node $(k, a_{d-1}\cdots a_1a_0)$ to destination $(l, b_{d-1}\cdots b_1b_0)$, incorporating the resilient connection pattern of Cycloid. The routing algorithm as shown in Fig. 3, involves three phases, assuming MSDB be the most significant different bit of the current node and the destination:

(1) *Ascending*: When a node receives a request, if its $k <$ MSDB, it forwards the request to a node in the outside leaf set sequentially until cyclic index $k \geq$ MSDB.
(2) *Descending*: In the case of $k \geq$ MSDB, when $k =$ MSDB, the request is forwarded to the cubical neighbor, otherwise the request is forwarded to the cyclic neighbor or inside leaf set node, whichever is closer to the target, in order to change the cubical index to the target cubical index.
(3) *Traverse cycle*: If the target ID is within the leaf sets, the request is forwarded to the closest node in the leaf sets until the closest node is the current node itself.

Specifically, the ascending phase selects the node whose cubical index is numerically closest to the destination out of the outside leaf set. In the descending phase, when $k >$ MSDB, the node whose cubical index is the closest to the destination and cyclic index $\leq$ MSDB is chosen among the cyclic neighbors and the inside leaf set. When choosing node from the cyclic neighbors, the cyclic neighbor that is closer to the destination should be chosen. That is, if the key's cubical index is toward the clockwise direction of the present node, the larger cyclic neighbor should be chosen, otherwise the smaller cyclic neighbor should be chosen. As a result, the $a_{d-1}\cdots a_1a_0$ might be changed to $a'_{d-1}\cdots a'_1a'_0$. When $k =$ MSDB, the prefix routing algorithm is used in the left-to-right order as Pastry [17] to change $a_{d-1}\cdots a_1a_0$ to $b_{d-1}\cdots b_1b_0$. Along each cubical link, the message is sent to the next node that shares at least one more bit in prefix with the key than the current node. After a cubical link is taken, one or more cyclic links or inside leaf set are taken in order to change the cyclic index to the MSDB between the current node and the key. Thus, cubical and cyclic links or inside leaf set are used alternately. In the traverse cycle phase, if the destination is within the local cycle, the message is sent to the a node in the inside leaf set, otherwise it is sent to a node in the outside leaf set, whichever is numerically closer to the destination. If the nearest node is itself, then the destination is reached and the search is completed. During the entire lookup process, if a node cannot be found or if a node has failed, the node that is numerically closer to the destination among the leaf sets is chosen.

Fig. 4 presents an example of routing a request from node (0,0100) to node (2,1111) in a four-dimensional Cycloid DHT. The MSDB of node (0,0100) with the destination is 3. As (0,0100) cyclic index $k = 0$ and $k <$ MSDB, it is in the ascending phase. Thus, the node (3,0010) in the outside leaf set is chosen. Node (3,0010)'s cyclic index 3 is equal to its MSDB, then in the descending phase, the request is forwarded to its cubical neighbor (2,1010). After node (2,1010) finds that its cyclic index is equal to

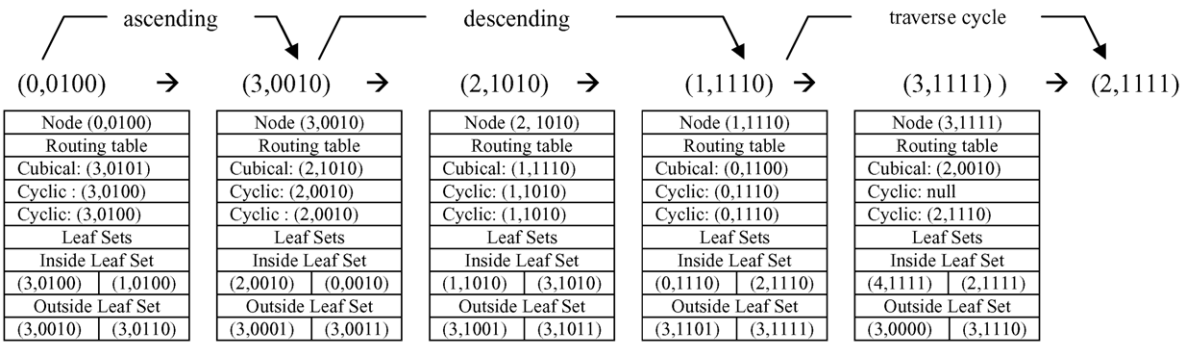

Fig. 4. An example of routing phases and routing table states in Cycloid.

its MSDB 2, it forwards the request to its cubical neighbor (1,1110). Because the destination (2,1111) is within its leaf sets (1,1110) forwards the request to the closest node to the destination (3,1111). Similarly, after (3,1111) finds that the destination is within its leaf sets, it forwards the request to (2,1111) and the destination is reached.

Each of the three phases is bounded by O(d) hops, hence the total path length is O(d). The key idea behind this algorithm is to keep the distance decrease repeatedly. In [4], we proved the correctness of the routing algorithm by showing its convergence and reachability. By convergence, we mean that each routing step reduces the distance to the destination. By reachability, we mean that each succeeding node can forward the message to the next node. Because each step sends the lookup request to a node that either shares a longer prefix with the destination than the current node, or shares as long a prefix with, but is numerically closer to the destination than the current node, the routing algorithm is convergent. Also, the routing algorithm can be easily augmented to increase fault tolerance. When the cubical or the cyclic link is empty or faulty, the message can be forwarded to a node in the leaf sets.

Our discussion so far is based on a seven-entry Cycloid DHT. It can be extended to include two predecessors and two successors in its inside leaf set and outside leaf set, respectively. We will show via simulations in the next section that the 11-entry Cycloid DHT has better performance.

3.3. Self-organization

Peer-to-peer systems are notoriously dynamic in the sense that nodes are frequently joining in and departing from the network. Cycloid deals with node joining and leaving in a distributed manner, without requiring hash information to be propagated through the entire network. This section describes how Cycloid handles node joining and leaving.

3.3.1. Node join

When a new node joins, it first gets its ID by using the method introduced in Section 3.1. It then initializes its routing table and leaf sets, and informs other related nodes of its presence. Like Chord [19] and Viceroy [12], Cycloid assumes that any new node initially knows about a live node. Assume the first contact node is $A = (k, a_{d-1}a_{d-2}\cdots a_0)$ and the new node is $X = (l, b_{d-1}b_{d-2}\cdots b_0)$. According to the routing algorithm discussed in Section 3.2, the node A will route the joining message to the existing node Z whose ID is numerically closest to the ID of X. Z's Leaf Sets are the basis for X's Leaf Sets. In

particular, the following two cases are considered:

(1) If X and Z are in the same cycle, Z's outside leaf set becomes the X's outside leaf set. X's inside leaf set is initiated according to Z's inside leaf set. If Z is X's successor, Z's predecessor and Z are the left node and right node in X's inside leaf set respectively. Otherwise, Z and Z's successor are the left node and right node.
(2) If X is the only node in its local cycle, then Z is not in the same cycle as X. In this case, two nodes in X's inside leaf set are X itself. X's outside leaf set is initiated according to Z's outside leaf set. If Z's cycle is the succeeding remote cycle of the X, Z's left outside leaf set node and the primary node in Z's cycle are the left node and right node in X's outside leaf set. Otherwise, the primary node in Z's cycle and Z's right outside leaf set node are the left node and right node in X's outside leaf set.

We use a local–remote method to initialize the three neighbors in the X's routing table. It searches for a neighbor in the local cycle in a decreasing order of the node cyclic index. If the neighbor is not found, then its neighboring remote cycle is searched. The remote cycle search sequence depends on the kth bit in the cubical index. If a_k is 1, the search direction is counter-clockwise, otherwise the direction is clockwise. This is done in order to enhance the possibility and the speed of finding the neighbors.

After a node joins the system, it needs to notify the nodes in its inside leaf set. It also needs to notify the nodes in its outside leaf set if it is the primary node of its local cycle. Once the nodes in the inside leaf set receive the joining message, they will update themselves. When the nodes in the outside leaf set receive the joining message, in addition to update themselves, they need to transfer the message to the nodes in their inside leaf set. Thus, the message is passed along in the joining node's neighboring remote cycle until all the nodes in that cycle finish updating.

3.3.2. Node departure

Before a node leaves, it needs to notify its inside leaf set nodes. In Cycloid, a node only has outgoing connections and has no incoming connections. Therefore, 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. Upon receiving a leaving notification, the nodes in the inside and outside leaf sets update themselves. In addition, the nodes in the outside leaf set need to notify other nodes in their local cycle one by one, which will take at most d steps. As a result, only those who take the leaving node as their inside leaf set or outside leaf set are updated. Those nodes who take the leaving node as their cubical neighbor or cyclic neighbor cannot be updated. Updating cubical and cyclic neighbors are the responsibility of system stabilization, as in Chord.

3.3.3. Fault tolerance

Undoubtedly, low degree P2P networks perform poorly in failure-prone environments, where nodes fail or depart without warning. Usually, the system maintains another list of nodes to handle such problems, such as the successor list in Chord [19] and the bucket in Viceroy [12]. In this paper, we assume that nodes must notify others before leaving, as the authors of Koorde argued that the fault tolerance issue should be handled separately from routing design.

4. Cycloid performance evaluation

In [9], Kaashoek and Karger listed five primary performance measures of DHTs: degree in terms of the number of neighbors to be connected, hop count per lookup request, degree of load balance, degree of fault tolerance, and maintenance overhead. In this section, we evaluate Cycloid in terms of these performance measures and compare it with other two constant-degree DHTs: Viceroy and Koorde. Recall that each Cycloid node maintains connectivity to seven neighbors in its routing table. Cycloid can be extended to include more predecessors and successors in its inside and outside leaf sets for a trade-off for lookup hop count. The results due to 11-neighbor Cycloid are included for a demonstration of the trade-off. Similarly, Koorde DHT provides a flexibility to making a trade-off between routing table size and routing hop count. For a fair comparison, in our simulations, we assumed the Koorde DHT maintained connectivity to seven neighbors, including one de Bruijn node, three successors and three immediate predecessors of the de Bruijn node. Since all of the constant-degree DHTs borrowed ideas from Chord and other DHTs with O(log n) neighbors, we also include the results of Chord as references. The actual number of participants varied in different experiments. We designed and implemented simulators in Java for the DHTs. The simulation results due to Chord, Koorde, and Viceroy were verified with those in their original publications.

4.1. Key location efficiency

It is known that all of the constant-degree DHTs have a complexity of O(log n) or O(d) hops per lookup request with O(1) neighbors. Although Cycloid contains more nodes than the others for the same network dimension, its average routing performance relative to Viceroy and Koorde is unknown. In this experiment, we simulated networks with $n = d \times 2^d$ nodes and varied the dimension d from 3 to 8. Each node made a total of $n/4$ lookup requests to random destinations. Figs. 5 and 6 plot the mean of the measured path lengths of the lookup requests due to various DHT routing algorithms with respect to the network size and dimension, respectively. The path length of each request is measured by the number of hops traversed during its search.

From Fig. 5, we can see that the path lengths of Viceroy are more than two times than those of Cycloid, although key locations in both Cycloid and Viceroy involve the same ascending, descending and traverse ring/cycle phases. There are two reasons. First, the ascending phase in Cycloid usually takes only one

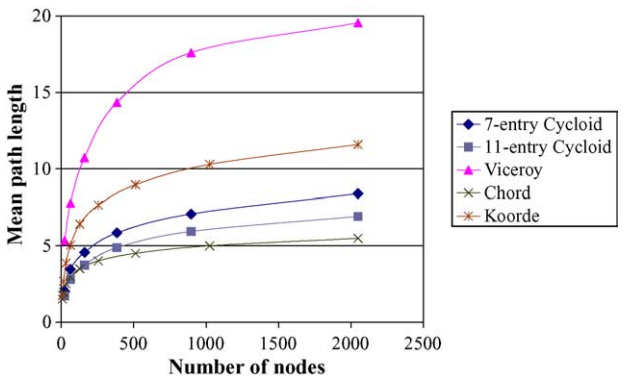

Fig. 5. Path lengths of lookup requests in various DHTs of different network sizes.

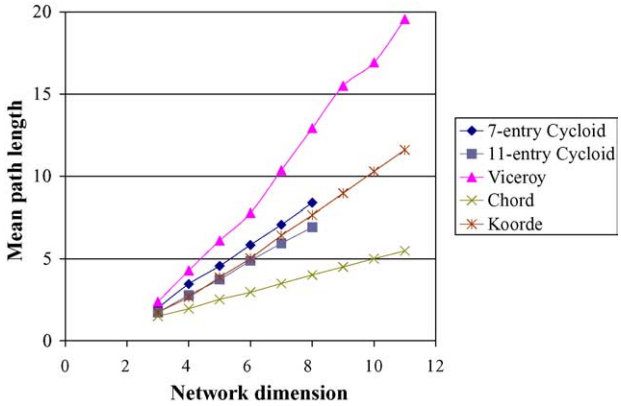

Fig. 6. Path lengths as a function of network dimension in various DHTs.

step because the outside leaf set entry node is the primary node in its cycle. But the ascending phase in Viceroy takes $(\log n)/2$ steps on average because each step decreases the level one at a time. Fig. 7(a) and (b) present breakdowns of the lookup cost in different phases in Cycloid and Viceroy, respectively. From the figures, we can see that the ascending phase in Viceroy constitutes about 30% of the total path length, but only up to 15% in Cycloid. Second, the descending phase in Cycloid takes d steps because each step redirects the request to a node with longer prefix or is numerically closer to the target. It is followed by another d hops of search in local cycles or cubic neighbor cycles. In Viceroy, the distance to the target can be halved each step in the second descending phase. But Fig. 7(b) shows that the descending phases constitute around only 20% of the total searching path. More than half of the cost is spent in the third traverse ring phase. In the traverse ring phase, the lookup request approaches the destination step by step along ring links or level ring links and needs another $(\log n)/2$ steps on average.

In Koorde, each node redirects an incoming lookup request to its first de Bruijn node or a successor. Each selection of a first de Bruijn node would reduce the distance by half. Since the first de Bruijn node may not be the immediate predecessor of the imaginary node of the destination, selection of a successor is to find the immediate predecessor. Fig. 7(c) shows a breakdown of the cost between the two selections. The selection of successors constitutes about 30% of the total path length, which implies some nodes might interpose land in between the current node's first de Bruijn node and the imaginary node. In this case, the current node's successors have to be passed in order to reach the immediate predecessor of the imaginary node. Because of the dense network in which every node is alive, there are only a few nodes at interpose between de Bruijn node and the imaginary node, consequently, the path length of taking successors takes a reasonable percentage of the whole path length. However, Koorde's lookup efficiency is reduced in sparse network. We will discuss this in Section 4.5.

The principle of Cycloid routing algorithm is almost the same as that of Koorde. In both algorithms, starting from a specific chosen node, the node ID bits are changed one by one until the target node ID is reached. Both of their path lengths are close to d, the dimension of the network in simulation. Since a d-dimensional Cycloid contains more $(d-1)\times 2^d$ nodes than Koorde of the same dimension, Cycloid leads to shorter lookup path length than Koorde in networks of the same size, as shown in Fig. 6.

From Fig. 6, we can also see that the path length of Viceroy increases much faster with respect to the network dimension. Its path length increases from 4 in a four-dimensional network to 12.91 in an

eight-dimensional network. It is because an increase of one dimension in Cycloid contains $(d + 1)$ times more nodes than in Viceroy and Koorde. That is, Cycloid leads to a smaller dimension for large-scale networks and higher key location efficiency.

4.2. Load balance

A challenge in the design of balanced DHTs is to distribute keys evenly between a changing set of nodes and to ensure each node experiences even load as an intermediate for lookup requests from other nodes. Cycloid deals with the key distribution problem in a similar way to Koorde, except that Cycloid uses a pair of cyclic and cubical indices to represent a node. Viceroy maintains a one-dimensional ID space. Although both Cycloid and Viceroy nodes have two indices to represent their place in the overlay network, the cyclic index is part of the Cycloid node ID but the level is not part of the Viceroy node ID. Also, Viceroy stores keys in the keys' successors. Table 3 shows the differences in key assignment between the DHTs.

In this experiment, we simulated different DHT networks of 2000 nodes each. We varied the total number of keys to be distributed from 10^4 to 10^5 in increments of 10^4. Fig. 8 plots the mean, the 1st

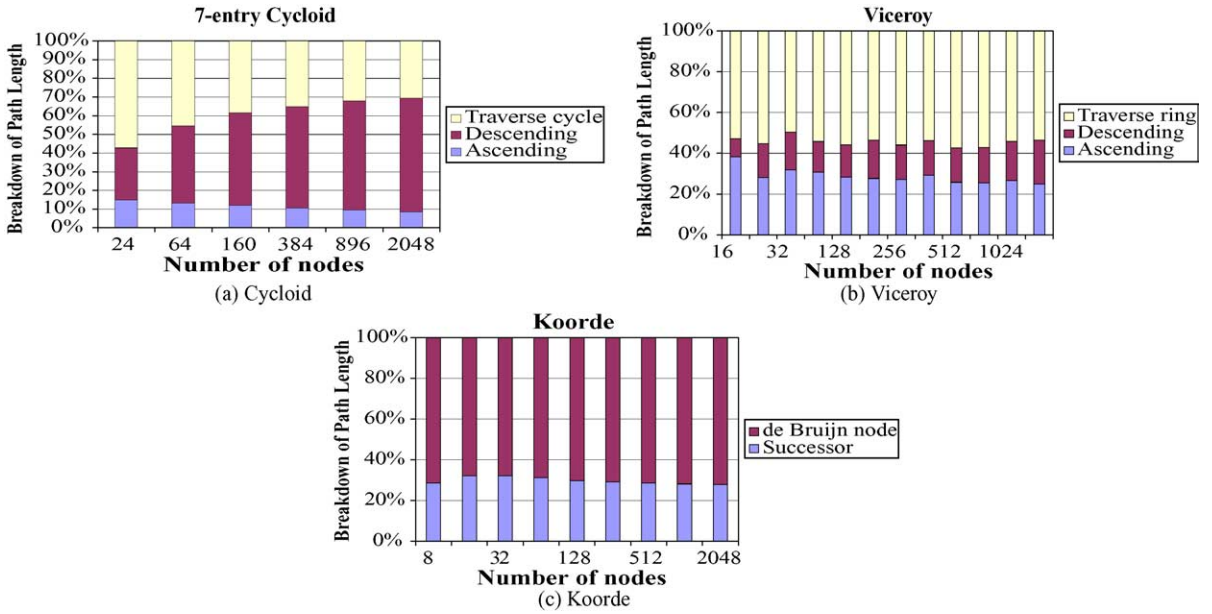

Fig. 7. Path length breakdown in various DHTs of different sizes.

Table 3
Characterization of node identification and key assignment in different DHTs

	Cycloid	Viceroy	Koorde
Base network	CCC	Butterfly	de Bruijn
ID space	$([0, d), [0, d \times 2^d))$	$([0, 3 \log n), [0, 1))$	$[0, 2^d)$
Node identity	$(k, a_{d-1}a_{d-2} \cdots a_0)$ k is static	(level, ID) level is dynamic	Id
Key placement	Numerically closest node	Successor	Successor

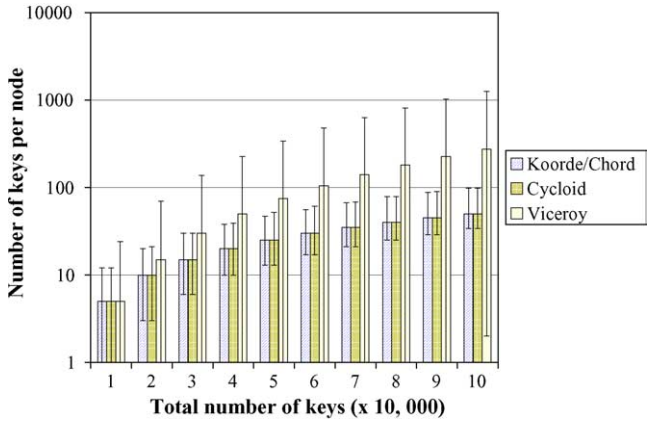

Fig. 8. Key distribution in networks of 2000 nodes due to different DHTs. Assume the network ID space is of 2048 nodes.

and 99th percentiles of the number of assigned keys per node when the network ID space is of 2048 nodes. The number of keys per node exhibits variations that increase linearly with the number of keys in all DHTs. The key distribution in Cycloid has almost the same degree of load balance as in Koorde and Chord because Cycloid's two-dimensional ID space is reduced to one-dimension by the use of a pair of modula and divide operations. By comparison, the number of keys per node in Viceroy has much larger variations. Its poor balanced distribution is mainly due to the large span of real number ID space in [0, 1). In Viceroy, the key is stored in its successor; that is, a node manages all key-value between its counter-clockwise neighbor and itself. Because of Viceroy's large ID span, its node identifiers may not uniformly cover the entire space, some nodes may manage much more keys than the others.

Fig. 9 plots the mean, the 1st and 99th percentiles of the number of keys per node in Cycloid and Koorde DHTs when there are only 1000 participants in the network. From the figure, it can be seen that Cycloid leads to a more balanced key distribution than Koorde for a sparse network. In Koorde, the node identifiers do not uniformly cover the entire identifier space, leading to unbalanced key allocation as a result of storing the key in its successor. By comparison, using two-dimension key allocation method,

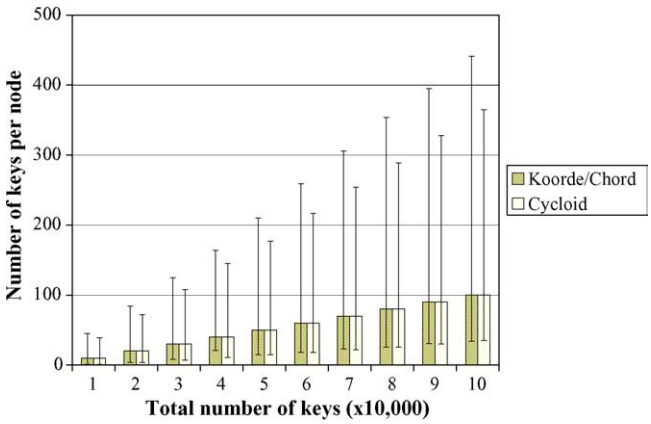

Fig. 9. Key distribution in networks of 1000 nodes due to different DHTs. Assume the network ID space is of 2048 nodes.

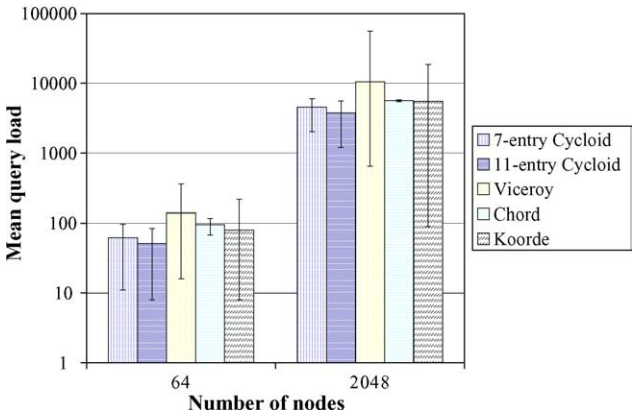

Fig. 10. Query load variances in various DHTs of different sizes.

Cycloid achieves better load balance by storing the key in its numerically closest node; that is, the keys between a node's counter-clockwise neighbor and itself will be allocated to that neighbor or the node itself rather than to itself totally. Chord solved this problem by replicating each node into O(log n) "virtual nodes", but such replication would destroy the optimality of constant degree in Koorde. In [9], Kaashoek and Karger put forward a question of finding a system that is both degree optimal and load balanced. Cycloid should be an answer.

In summary, when the entire identifier space is mostly occupied, Cycloid's load balance is as good as Chord. When the actual nodes only occupy small part of the total entire identifier space, Cycloid's load balance is better than Chord.

Key distribution aside, another objective of load balancing is to balance the query load between the participating nodes. The query load is measured as the number of queries received by a node for lookup requests from different nodes.

Fig. 10 plots the mean, the 1st and 99th percentiles of query loads of various DHT networks of 64 and 2048 nodes. The figures shows that Cycloid exhibits the smallest variation of the query load, in comparison with other constant-degree DHTs. This is partly due to the symmetric routing algorithm of Cycloid.

In Viceroy, the ascending phase consists of a climb using up connections until level 1 is reached and the descending phase routes down the levels of tree using the down links until no down links exist. As a result, the nodes in the higher levels will be the hot spots, on the other hand, the nodes of the lower levels have smaller workload, which leads to the great workload variation, especially in the large-scale network. In Koorde, the first de Bruijn of a node with ID m is the node immediately precedes $2m$; that is, the predecessor of the ID got by shitting m to the left one bit. So, all the first de Bruijn nodes' identifiers are even in a "complete" (dense) network and with high probability the IDs are even in an incomplete (sparse) network. Consequently, the nodes with even IDs have heavy workload while nodes with odd IDs have light workload according to the lookup algorithm of Koorde. In Cycloid, because of the leaf sets, the nodes with small cyclic index, typically 0 or 1, will be light loaded. However, these nodes constitute only small part of the Cycloid network, in comparison with the hot spots in Viceroy and Koorde. Cycloid performs better than Viceroy and Koorde in the aspect of congestion.

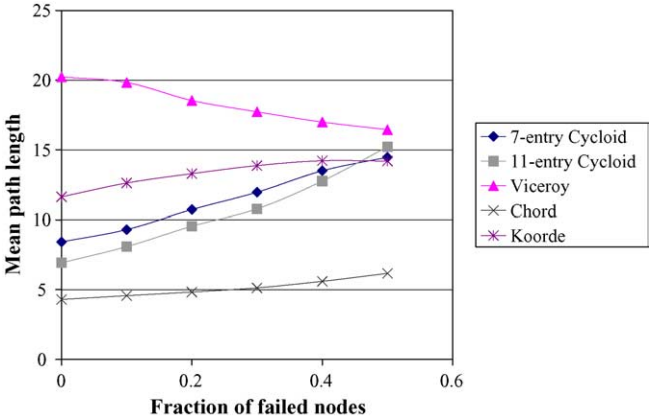

Fig. 11. Path lengths of various DHTs with different nodal departure probabilities.

4.3. Simultaneous node failures/departures

In this section, we evaluate the impact of massive node failures and/or departures on the performance of various DHTs, and on their capability to performing correct lookups without stabilization. We use the term of departure to refer to both failure and departure. We assume that node departures are graceful; that is, a node informs its relatives before its departure. Ungraceful departure is not discussed in this paper.

In this experiment, we simulated a network of 2048 nodes. Once the network becomes stable, each node is made to fail with probability p ranging from 0.1 to 0.5. After a failure occurs, we performed 10,000 lookups with random sources and destinations. We recorded the number of timeouts occurred in each lookup, the lookup path length, and whether the lookup found the key's correct storing node. A timeout occurs when a node tries to contact a departed node. The number of timeouts experienced by a lookup is equal to the number of departed nodes encountered. Fig. 11 shows the mean path length of the lookups with the change of departure probability p in different DHTs. The mean, the 1st and 99th percentiles of the number of timeouts of each DHTs are presented in Table 4.

In Cycloid, the path length increases due to the increasing of the number of timeouts as the p increases. Recall that when a departed node is met, the leaf sets have to be turned to for the next node. Therefore, the path length increases. All lookups were successfully resolved means that the Cycloid is robust and reliable.

We can see from Fig. 11 that unlike Cycloid and Koorde, the lookup path length in Viceroy decreases with the increase of p. In Viceroy, a node has both outgoing and incoming connections. A node notifies its

Table 4
Number of timeouts in different DHTs as more nodes depart/fail

Departure probability	Seven-entry Cycloid	Eleven-entry Cycloid	Viceroy	Chord	Koorde
0.1	0.53 (0, 4)	0.56 (0, 4)	0 (0, 0)	0.62 (0, 6)	0.02 (0, 1)
0.2	1.24 (0, 8)	1.38 (0, 8)	0 (0, 0)	1.37 (0, 8)	0.04 (0, 1)
0.3	2.46 (0, 11)	2.14 (0, 10)	0 (0, 0)	2.38 (0, 11)	0.06 (0, 2)
0.4	4.09 (0, 17)	3.67 (0, 15)	0 (0, 0)	3.91 (0, 16)	0.08 (0, 3)
0.5	5.88 (0, 24)	5.24 (0, 25)	0 (0, 0)	6.53 (0, 26)	0.09 (0, 4)

outgoing and incoming connections before its departure. Therefore, all related nodes are updated before the node departs. Based on this characteristic, massive departures have no adverse effect on Viceroy's ability to perform correct lookups. Table 4 shows that Viceroy has no timeouts. The decrease of the path length is caused by the decrease of the network size. We can see from Fig. 11 that when the departure probability is 0.5, the path length is 16.45, which is very close to the 1024 nodes complete network lookup's average path length 16.92 in Fig. 5.

In order to eliminate the impact of simultaneous node departures in Viceroy, a leaving node would induce O(log n) hops and require O(1) nodes to change their states. This causes a large amount of overhead. In Cycloid, the path length increased a little with a small fraction of departed nodes. Even though the path length of Cycloid increases slightly, it is still much less than that of Viceroy.

In Fig. 11, Koorde's path length increased not so much as in Cycloid when the node departure probability p exceeds 0.3. Unlike Cycloid and Viceroy, Koorde has lookup failures when p becomes larger than 0.3. Our experiment results show that there are 791, 1226, and 4259 lookup failures when $p = 0.3$, 0.4, and 0.5, respectively.

In Koorde, when a node leaves, it notifies its successors and predecessor. Then, its predecessor will point to its successor and its successor will point to its predecessor. By this way, the ring consistency is maintained. The nodes who take the leaving node as their first de Bruijn node or their first de Bruijn node's predecessor will not be notified and their update are the responsibility of stabilization.

Each Koorde node has three predecessors of its first de Bruijn node as its backups. When the first de Bruijn node and its backups have all failed, the Koorde node fails to find the next node and the lookup fails. When the failed node percentage is as low as 0.2, all the queries can be solved successfully at a marginal cost of query length with increase path length as shown in Fig. 11. When p exceeds 0.3, with increasing timeouts as shown in Table 4, the number of failures increase, and the path length increases not so much as before because less backups are taken.

From Table 4, we can see that although Koorde has much less timeouts than Cycloid, it still has a large number of failures. In Koorde, the critical node in routing is the de Bruijn node whose backups cannot always be updated. In contrast, Cycloid relies on updated leaf sets of each node for backup. Therefore, Koorde is not as robust as Cycloid in response to massive node failures/departures. The experiment shows that Cycloid is efficient in handling massive node failures/departures without stabilization.

4.4. Lookups during node joining and leaving

In practice, the network needs to deal with nodes joining the system and with nodes that leave voluntarily. In this paper, we assume that multiple join and leave operations do not overlap. We refer the reader to [11] for techniques to achieve concurrency and to handle failures in the system. In this experiment, we compare Cycloid with Viceroy and Koorde when nodes join and leave continuously.

The setting of this experiment is exactly the same as the one in [19]. Key lookups are generated according to a Poisson process at a rate of one per second. Joins and voluntary leaves are modeled by a Poisson process with a mean rate of R, which ranges from 0.05 to 0.40. A rate of $R = 0.05$ corresponds to one node joining and leaving every 20 s on average. In Cycloid, each node invokes the stabilization protocol once every 30 s and each node's stabilization routine is at intervals that are uniformly distributed in the 30 s interval. Thus, R ranges from a rate of 1.5 joins and 1.5 leaves per one stabilization periods to a rate of 12 joins and 12 leaves per one stabilization period. The network starts with 2048 nodes.

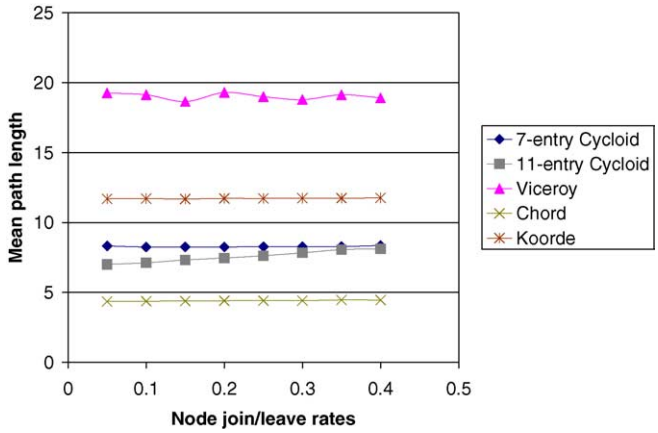

Fig. 12. Path lengths of lookup requests in different DHTs as the node join/leave rates change.

Fig. 12 shows the mean path length of lookup operations in different DHTs as the node join/leave rate R changes. The mean, the 1st and 99th percentiles of the number of timeouts are shown in Table 5. There are no failures in all test cases. From the path length evaluation in Section 4.1, we know that the mean path length of Cycloid in steady states is 8.38. From Fig. 12, we can see that the measured path lengths in the presence of node joining and/or leaving are very close to this value and do not change with the rate R. This is because with the help of stabilization, there are less needs for a node to turn to its leaf sets in the case of meeting an absent or departed node. Consequently, a lookup request would experience less timeouts and its path length remains unchanged. Compared with the timeout results in Table 4, we can see that stabilization removes majorities of the timeouts.

In Koorde, the path lengths changed little compared to 11.59 in stable network though the timeouts increase with the rate of node joins and leaves. The failure time is reduced to 0 compared to the large failure time in Section 4.3. That's because stabilization updates the first de Bruijn node of each node and the de Bruijn node's predecessors in time. When the first de Bruijn node and its predecessors have all failed, then passed lookups will fail with high probability.

Table 5
Number of timeouts as the node join/leave rate changes

Node join/leave rate (per second/per stable period)	Seven-entry Cycloid	Eleven-entry Cycloid	Viceroy	Chord	Koorde
0.05/1.5	0.005 (0, 0)	0.059 (0, 2)	0 (0, 0)	0.033 (0, 1)	0.003 (0, 0)
0.10/3	0.009 (0, 0)	0.103 (0, 2)	0 (0, 0)	0.078 (0, 2)	0.013 (0, 1)
0.15/4.5	0.014 (0, 1)	0.171 (0, 2)	0 (0, 0)	0.130 (0, 2)	0.008 (0, 0)
0.20/6	0.031 (0, 1)	0.205 (0, 3)	0 (0, 0)	0.125 (0, 2)	0.013 (0, 1)
0.25/7.5	0.047 (0, 2)	0.246 (0, 2)	0 (0, 0)	0.151 (0, 2)	0.016 (0, 1)
0.30/9	0.052 (0, 2)	0.289 (0, 3)	0 (0, 0)	0.191 (0, 3)	0.016 (0, 1)
0.35/10.5	0.058 (0, 2)	0.367 (0, 4)	0 (0, 0)	0.220 (0, 3)	0.023 (0, 1)
0.40/12	0.070 (0, 2)	0.374 (0, 4)	0 (0, 0)	0.233 (0, 3)	0.023 (0, 1)

The results show that Viceroy's performance is not affected by the node leaving and joining. That's because, before a node leaves and after a node joins, all the related nodes are updated. Although Viceroy has no timeouts, its path length is much longer compared to Cycloid's path.

Though Viceroy does not run stabilization periodically, it needs to update all related node for node joining and leaving. Therefore, it is not suitable for a dynamic network with frequent node arrivals and departures considering the join and leave high cost.

4.5. Impact of network sparsity in the ID space

Due to the dynamic nature of peer-to-peer systems, a DHT needs to maintain its location efficiency, regardless of the actual number of participants it has. But, in most of the DHTs, some node routing table entries are void when not all nodes are present in the ID space. For example, if a local cycle in Cycloid has only one node, then this node has no inside leaf set nodes. It is also possible that a node cannot find a cubical neighbor, or cyclic neighbor. We define the degree of sparsity as the percentage of non-existent nodes relative to the network size. To examine the impact of sparsity on the performance of other systems, we did an experiment to measure the mean search path length and the number of failures when a certain percentage of nodes are not present. We tested a total of 10,000 lookups in different DHT networks with an ID space of 2048 nodes. Fig. 13 shows the results as the degree of network sparsity changes. There are no lookup failures in each test case. From the figure, we can see that Cycloid keeps its location efficiency and the mean path length decreases slightly with the decrease of network size. In Viceroy, it's impossible for nodes to fully occupy its ID space because the node ID $\in [0, 1)$. Therefore, it is very likely that some links of a node are void and hence the sparsity imposes no effect on the location efficiency. In Koorde, the path length increases when the actual number of participants drops. This is because a sparse Koorde DHT exhibits a large span between two neighboring nodes. Since Koorde routes a lookup request through the immediate real predecessor of each imaginary node on the de Bruijn path, the distance between the imagination node and its immediate predecessor in the sparse DHT leads to a longer distance between the predecessor's first de Bruijn node and the imagination node in the next step. Therefore, more successors need to be taken to reach the immediate predecessor of the imagination, thus more path

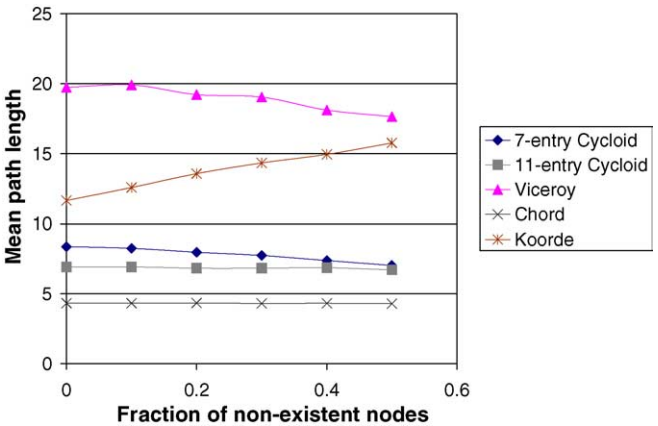

Fig. 13. Path length of lookup requests in different DHTs with the change of the degree of network sparsity.

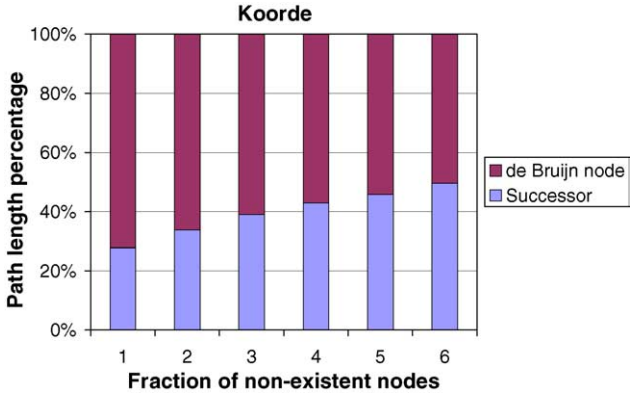

Fig. 14. Path length breakdown in Koorde with the change of the degree of network sparsity.

length. Fig. 14 presents a breakdown of the lookup cost of de Bruijn nodes and successors. From the figure, we can see that with the increase of sparsity, the lookup routing path involves more percentage of successors.

In summary, the sparsity does not have adverse effect on the location efficiency in Cycloid. However, Koorde's performance degrades with the decrease of the number of actual participants.

5. Conclusions

In this paper, we have presented a constant-degree DHT, namely Cycloid, and compared it with other two constant-degree DHTs: Viceroy and Koorde. Cycloid is based on Pastry and CCC, while Viceroy and Koorde emulate butterfly and de Bruijn graphs, respectively. Cycloid resembles Viceroy and Koorde in appearance because CCC is a subgraph of butterfly network and de Bruijn is a coset graph of butterfly, as recently proved in graph theories [3,8]. But they are different in connectivity maintenance of dynamic participants and in routing for key location.

We have evaluated the performance of the DHTs in terms of the lookup hop count, degree of load balance, degree of fault tolerance and cost for maintenance. Experiment results show that:

- Cycloid yields the best average-case location efficiency.
- Cycloid distributes keys and query load more evenly between the participants than Viceroy. In comparison with Koorde, Cycloid results in higher degrees of load balance for sparse networks and the same degree of load balance for dense networks.
- Cycloid is more robust because it continues to function correctly when a node's information is only partially correct. In contract, Koorde cannot perform well with partially correct information and incomplete network.
- Cycloid scales well with the number of nodes, recovers from large numbers of simultaneous node departures and joins, and answers lookups correctly even during recovery. By contrast, Viceroy handles massive node failures/departures at a high cost for connectivity maintenance, especially in the case when a node needs to change its level.

A common problem with constant-degree DHTs is their weakness in handling node leaving without warning in advance. Keeping more information like successor list in Chord [19] and Bucket in Viceroy [12] helps to resolve the problem, but destroys the optimality of constant degree. Because of this disadvantage, whenever a node joins or leaves, Cycloid needs to notify its inside leaf set. Especially, if the joining or leaving node is the primary node of a cycle in Cycloid, the updates might produce much more overhead. In addition, the initialization and updates of three neighbors in the routing table also might cause a lot of overhead. These issues need to be addressed for a refinement of Cycloid.

Acknowledgments

This research was supported in part by U.S. NSF grants ACI-0203592. Guihai Chen's work was partly supported by China 973 project (No. 2002CB312002) and China NSF Grant (No. 60073029). We thank the reviewers for constructive comments and suggestions.

References

[1] Freenet. http://freenet.sourceforge.net.
[2] Gnutella. http://www.gnutella.com.
[3] F. Annexstein, M. Baumslag, A.L. Rosenberg, Group action graphs and parallel architecture, SIAM J. Comput. 19 (1990) 544–569.
[4] G. Chen, C. Xu, H. Shen, P2P overlay networks of constant degree, in: Proceedings of the International Workshop on Grid and Cooperative Computing, Lecture Notes in Computer Science, vol. 2975, 2003.
[5] I. Clarke, A distributed decentralised information storage and retrieval system, Technical Report, Division of Informatics, University of Edinburgh, UK, 1999.
[6] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong, Freenet: a distributed anonymous information storage and retrieval system, in: Proceedings of the ICSI Workshop on Design Issues in Anonymity and Un-observability, 2000.
[7] Clip2. com. The Gnutella protocol specification v0.4., 2000. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.
[8] R. Feldmann, W. Unger, The cube-connected cycles network is a subgraph of the butterfly network, Parallel Process. Lett. 2 (1) (1991) 13–19.
[9] M.F. Kaashoek, R. Karger, Koorde: a simple degree-optimal distributed hash table, in: Proceedings of the Second International Workshop on P2P Systems (IPIPS), 2003.
[10] Q. Lv, S. Shenker, Search and replication in unstructured peer-to-peer networks, in: Proceedings of ACM SIGGRAPH, August 2002.
[11] N. Lynch, D. Malkhi, D. Ratajczak, Atomic data access in distributed hash tables, in: Proceedings of the International Peer-to-Peer Symposium, 2002.
[12] D. Malkhi, M. Naor, D. Ratajczak, Viceroy: a Scalable and Dynamic Emulation of the Butterfly, in: Proceedings of Principles of Distributed Computing (PODC'02), 2002.
[13] P. Maymounkov, D. Mazires, Kademlia: A peer-to-peer information system based on the xor metric, in: Proceedings of the 1st Interational Workshop on Peer-to-Peer Systems (IPTPS), 2002.
[14] C. Plaxton, R. Rajaraman, A. Richa, Accessing nearby copies of replicated objects in a distributed environment, in: Proceedings of ACM SPAA, 1997, pp. 311–320.
[15] F.P. Preparata, J. Vuillemin, The cube-connected cycles: A versatile network for parallel computation, Commun. ACM 24 (5) (1981) 300–309.
[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of ACM SIGCOMM, 2001, pp. 329–350.

[17] A. Rowstron, P. Druschel, Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems, in: Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001.
[18] H. Shen, A. Brodie, C. Xu, W. Shi, Scalable and secure P2P overlay networks, in: J. Wu (Ed.), Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, CRC Press, Boca Raton, FL, 2004.
[19] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for Internet applications, IEEE/ACM Trans. Networking 11 (1) (2003) 17–32.
[20] J. Xu, A. Kumar, X. Yu, On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks, IEEE J. Sel. Area Commun. 22 (1) (2003) 151–163.
[21] B. Yang, H. Garcia-Molina, Comparing hybrid peer-to-peer systems, in: Proceedings of the 27th International Conference on Very Large Databases, 2001.
[22] B. Zhao, J. Kubiatowicz, A. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, Technical Report UCB/CSD-01-1141, Computer Science Division, UC Berkeley, 2001.

Haiying Shen received the BS degree in computer science from Tongji University, China in 2000, and the MS degree in computer engineering from Wayne State University in 2004. She is now a PhD student in the Department of Electrical and Computer Engineering at Wayne State University. Her research interests include P2P overlay networks, distributed and parallel computer systems.

Cheng-Zhong Xu received the BS and MS degrees from Nanjing University, China, in 1986 and 1989, respectively, and the PhD degree from the University of Hong Kong, China, in 1993, all in computer science. He is currently an associate professor in the Department of Electrical and Computer Engineer of Wayne State University (Detroit, MI). Dr. Xu has published more than seventy peer-reviewed papers in international journals and refereed conference proceedings in the areas of resource management in distributed and parallel systems, high performance cluster computing, mobile code, and scalable and secure Internet services. He is the co-author of the book "Load Balancing in Parallel Computers: Theory and Practice" (Kluwer Academic, 1997). He was a guest co-editor of Journal of Parallel and Distributed Computing on "Scalable Internet Services and Architectures" in October 2003. He has also served on technical program committees of numerous international conferences, including the recent IEEE ICDCS'04 and ICPP'04. His research was partly supported by U.S. NSF and NASA. He is a recipient of "Faculty Research Award" of Wayne State University in 2000, "President's Awards for Excellence in Teaching" in 2002, and "Career Development Chair Award" in 2003. He is a senior member of IEEE and a member of ACM.

Guihai Chen obtained BS degree from Nanjing University, M. Engineering from Southeast University, and PhD from University of Hong Kong. He visited Kyushu Institute of Technology, Japan in 1998 as a research fellow, and University of Queensland, Australia in 2000 as a visiting professor. During September 2001 to August 2003, he was a visiting professor in Wayne State University. He is now a full professor of Nanjing University. His research interests include interconnection networks, high-performance computer architecture, graph theory, P2P computing and network system.