# Leveraging a Compound Graph based DHT for Multi-Attribute Range Queries with Performance Analysis

Haiying Shen, * *Member, IEEE,* Cheng-Zhong Xu, *Member, Senior IEEE*

**Abstract**—Resource discovery is critical to the usability and accessibility of grid computing systems. Distributed Hash Table (DHT) has been applied to grid systems as a distributed mechanism for providing scalable range-query and multi-attribute resource discovery. Multi-DHT-based approaches depend on multiple DHT networks with each network responsible for a single attribute. Single-DHT-based approaches keep the resource information of all attributes in a single node. Both classes of approaches lead to high overhead. In this paper, we propose a Low-Overhead Range-query Multi-attribute DHT-based resource discovery approach (LORM). Unlike other DHT-based approaches, LORM relies on a single compound graph based DHT network and distributes resource information among nodes in balance by taking advantage of the compound graph structure. Moreover, it has high capability to handle the large-scale and dynamic characteristics of resources in grids. Experimental results demonstrate the efficiency of LORM in comparison with other resource discovery approaches. LORM dramatically reduces maintenance and resource discovery overhead. In addition, it yields significant improvements in resource location efficiency. We also analyze the performance of the LORM approach rigorously by comparing it with other multi-DHT-based and single-DHT-based approaches with respect to their overhead and efficiency. The analytical results are consistent with experimental results, and prove the superiority of the LORM approach in theory.

**Keywords: Multi-attribute range query, Distributed hash table, Peer-to-Peer, Resource discovery, Grids, Cycloid.**

## 1 Introduction

Grid systems integrate computers, clusters, storage systems and instruments to provide a highly available infrastructure for large scientific computing centers. Scalable and efficient resource discovery is critical to providing usability and accessibility in large-scale grid systems. The resources required by applications are often described by multi-attribute range queries. Such a query consists of a set of attributes such as available computing power and memory with a range for each attribute. A fundamental service of resource discovery is to locate resources according to the attribute inputs. Recently, Distributed Hash Table (DHT) architecture has been introduced to grid systems for large-scale and dynamic resource discovery due to its high scalability and dynamism-resilience features. DHT can efficiently route messages to the unique owner of any given object. Because of its single deterministic object location, the object can be either a resource attribute or a value. Therefore, it is a challenge to realize resource discovery with both range-query and multi-attribute requirements.

According to how to leverage the deterministic object location function of DHTs for multi-attribute range queries in grids, most current pure DHT based approaches can be classified into multi-DHT-based, single-DHT-based centralized and single-DHT-based decentralized approaches. Multi-DHT-based approaches support multi-attribute range queries by relying on multiple DHT networks with each network responsible for a single attribute [2, 5]. To locate resources specified by several attributes and ranges, each query for a resource is presented to the appropriate DHT and then the results are concatenated in a database-like join operation. However, maintaining multiple DHT networks generates a high cost, especially in a highly dynamic environment. Single-DHT-based centralized approaches [32] keep the resource information of all values for a specific attribute in a single node. In order to discovery resources satisfying user specified constraints, all nodes in the distributed system form a DHT and act as both resource servers and clients. The DHT server nodes collect the information of node resources, receive resource queries and send the requested resource information to the nodes. Though these approaches only need to maintain one DHT, it leads to high resource discovery latency since range searching is conducted in a very large resource information directory. Moreover, the centralized approach for an attribute may overload directory nodes by letting them maintain a high volume of information and process resource queries. Single-DHT-based decentralized approach [7, 8] is based on one DHT and provides range searching. It maps the attribute and value of a resource separately to one DHT, and processes a query by searching them separately. Though it incurs less DHT maintenance overhead and searching latency, it doubles the overhead for resource information maintenance, resource reporting and resource searching. This is because it separates each resource information piece to two parts in resource re-

- ** Corresponding Author. Email: shenh@clemson.edu.*
- *H. Shen is with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. C. Xu is with the Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202.*

porting and resource searching. In addition, it also accumulates resource information of a specific attribute in a single node, overloading these nodes for query processing.

To reduce overhead and enhance efficiency of resource discovery in grids, we propose a DHT-based resource discovery approach with features of Low-Overhead, Range-query and Multi-attribute (LORM) [40]. Unlike the aforementioned three groups of DHT-based approaches, LORM is built on a single DHT called Cycloid [41]. Instead of collecting resource information of all values of an attribute in a single node or separating each piece of resource information, LORM arranges each node to be responsible for the resource information of a specific attribute within a value range by taking advantage of the compound graph structure of Cycloid, which connects clusters by a cycle. Specifically, the resource information of one attribute is gathered in one cluster, and is further distributed among the nodes in the cluster according to the resource values. LORM achieves high efficiency through balanced distribution of maintenance overhead and resource discovery operations. We demonstrate its effectiveness and efficiency via simulation in comparison with other DHT-based approaches. We also analyze the performance of LORM rigorously and compare it with multi-DHT-based and single-DHT-based approaches with respect to their overhead and efficiency. The analytical results are consistent with the experimental results. The results prove the superiority of LORM through theory.

The proposed LORM approach can also be applied to other compound graph based overlays such as Kcube [18] and dBCube [10], in which clusters are connected by a structure such as Kautz digraph and de Bruijn graph. Then, for range-query and multi-attribute resource discovery, as in Cycloid, LORM can collect the resource information of one attribute in one cluster and further distribute the information among nodes in one cluster based on resource values.

The remainder of this paper is structured as follows. Section 2 describes a review of DHT-based resource discovery approaches for grid systems and methods for multidimensional and (or) range queries. Section 3 describes LORM, focusing on its resource discovery framework and algorithms. Section 4 presents the performance analysis of LORM in comparison to other representative DHT-based resource discovery methods. Section 5 analyzes the performance of LORM and other approaches with regards to the consistency between analytical results and experimental results in both a static environment and a dynamic environment. Section 6 concludes the paper and provides remarks on possible future work.

## 2 Related Work

Recently, numerous approaches have been proposed for resource discovery in grid systems. DHT [46, 25, 19, 20, 37, 58, 36, 41] has been widely adopted for resource discovery in grids. Current pure DHT based approaches to achieve multi-attribute range query can be generally classified into three groups: (1) *Multi-DHT-based approach* that adopts one DHT for each attribute, and processes multi-attribute range queries in parallel in corresponding DHTs [2, 5]. In this approach, multiple DHTs for multiple attributes need to be maintained, and the DHT key is used as the index for the resource value for range queries. However, the construction and maintenance of multiple DHTs are costly, especially in a dynamic environment. For example, suppose that there are $m$ types of resource attributes. Then, $m$ DHTs are needed. Although one node does not necessarily have all attributes, it is a member in each of the $m$ DHTs. The number of routing tables that a node maintains is $m$. Each routing table contains $\log n$ entries for a network with overall $n$ nodes. Therefore, each node needs to maintain $m \times \log n$ neighbors. (2) *Single-DHT-based centralized approach* that pools together resource information of all values for a specific resource attribute in a single node [32]. In this approach, the key in the DHT functions as the index for the resource attribute. However, this approach overloads directory nodes for maintaining high volume of resource information in their directories and resource query processing. In addition, a large directory size leads to inefficiency in resource searching. (3) *Single-DHT-based decentralized approach* that separately maps the resource attribute and value in the resource information to a single DHT, and processes a query by searching them separately [7, 8]. It brings about more overhead in resource reporting and searching, and produces more maintenance overhead for resource information by doubling the information pieces. In addition, it accumulates resource information of a specific attribute in a single node, which may lead to overloaded nodes.

There are other resource discovery methods for grids. The works in [45, 39] focus on multi-attribute resource discovery without considering range. They represent server meta-data as points in a multidimensional space and then express queries as predicates over these points. The works in [52, 47] leverage the tree structure for queries in grids. The works in [13, 31, 26] are based on unstructured P2P networks. However, tree-based approaches are not resilient to network churn and searching in unstructured networks generates a very high overhead.

There are many solutions to realize range queries and multidimensional queries over existing P2P systems. These methods can complement the pure DHT based approaches in structures, routing and data representation to achieve higher performance. ZNet [44], Midas [30], ERQ [56], SCAN [48] and SONAR [38] are multi-attribute range querying methods. Many other methods [27, 57, 17, 59, 29, 22] rely on a tree structure for this purpose. Shen and Li [43] and Zhang *et al.* [53] proposed two-layered architecture to support multidimensional range queries in P2P networks. Also, some methods [54, 6] use an index service or structure for multidimensional searching in P2P networks. Datta *et al.* [12] described how range queries can be supported in a structured overlay network that provides $O(\log n)$ search complexity on top of a trie abstraction. Wu *et al.* [51] proposed Roogle, a decentralized non-flooding P2P search engine that can efficiently support high-dimensional range queries in P2P systems. Ganesan *et al.* [16] showed how multidimensional queries may be supported in a P2P system by adapting traditional spatial database technologies (kd-trees and space-filling curves) with P2P routing networks and load-balancing algorithms. Chen *et al.* [11] exploited Bloom Filter for multi-keyword search over DHT P2P networks.

Also, there have been numerous studies on the problem of multidimensional or range queries in the database field. Early such works kd-tree [4], bdd-trees [3] and vp-tree [15]. Papadias *et al.* [33] showed how a variety of three- or four-dimensional queries that correspond to different relations can be processed using spatial data structures. In order to reduce the computing cost in dimensionality reduction while ensuring high query precision, Ravi *et al.* [23] proposed a technique that uses aggregate data from the existing index rather than the entire data. Castelli *et al.* [9] observed that a fundamental underpinning of any Content-based routing (CBR) protocol is for messages and subscriptions to "meet" at some points in the network, and thus proposed the HyperCBR which enforces this topological property in a multidimensional space. Weng *et al.* [49] proposed strategies to efficiently execute range queries on distributed machines, when partial replica optimization is employed.

LORM [40] is based on Cycloid [41]. It is advantageous over the three groups of DHT-based grid resource discovery methods in that it only relies on a single DHT to realize multi-attribute range-query resource discovery with low overhead. Moreover, unlike most other multidimensional range querying methods, LORM uses a DHT without the need of building an additional structure.

## 3 Range-Query and Multi-Attribute Resource Discovery

### 3.1 An Overview of Cycloid

This section provides an overview of DHT overlay networks followed by a high-level view of the Cycloid DHT. DHT overlays are a class of decentralized systems at the application level that partition ownership of a set of objects among participating nodes, and can efficiently route messages to the unique owner of any given object. Each object or node is assigned with an ID (i.e., key) that is the consistent hash value [24] of the object name or node IP address. The overlay network provides two main functions: `Insert(key,object)` and `Lookup(key)` that store an object to a node and retrieve the object, respectively. The message for the two functions is forwarded from node to node, based on the routing algorithm through the overlay network, until it reaches the object's owner. Each node maintains a routing table that records its neighbors in the overlay network.

Early studies have resulted in numerous DHT overlay networks. Our previous study has analyzed these overlays with various topologies and characteristics, and has designed the Cycloid overlay [41], which is based on a compound graph of *cube-connected cycles* [34]. Cycloid distinguishes itself from other DHTs by its higher scalability since its maintenance overhead is constant regardless of network size. Cycloid's low maintenance cost, balanced load distribution, and compound graph structure give LORM the capability of handling the challenges in resource discovery in grids. We present an overview of Cycloid in below.

Cycloid is a constant-degree overlay with $n{=}d{\cdot}2^d$ nodes, where $d$ is DHT dimension. It achieves a time complexity of $O(d)$ per lookup request by using $O(1)$ neighbors per node. Figure 1 shows a high-level structure of a 11-dimensional Cycloid, where $x{\in}[0, 10]$. The ID of each node or object in Cycloid is represented by a pair of indices $(k, a_{d-1}a_{d-2}\ldots a_0)$, where $k$ is called cyclic index and $a_{d-1}a_{d-2}......a_0$ is called cubical index. The cyclic index is an integer in $[0, d-1]$ and the cubical index is a binary number in $[0, 2^d-1]$. For example, in Figure 1, the ID of node $a$ is $(3, 200)$, i.e., $k=3$ and $a_{d-1}a_{d-2}\ldots a_0 = 200$ (in decimal value). For a given object or node, its cyclic index is its consistent hash value modulated by $d$ and its cubical index is its consistent hash value divided by $d$.

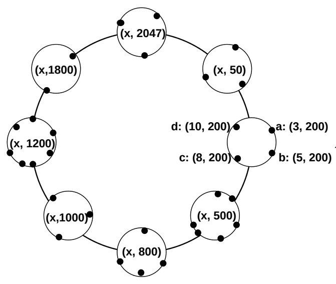


Fig. 1. Cycloid structure.

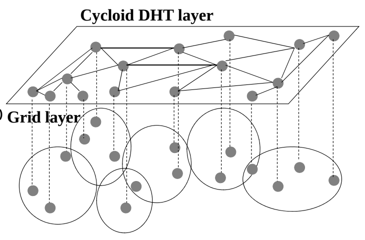


Fig. 2. LORM architecture.

As shown in Figure 1, the nodes with the same cubical index $a_{d-1}a_{d-2}......a_0$ are ordered by their $k$ on a small cycle, which we call *cluster*. The node with the largest $k$ in a cluster is called the *primary node* of the nodes at the cluster. All clusters are ordered by their cubical index $a_{d-1}a_{d-2}......a_0$ on a large cycle. For example, in the figure, nodes $a$, $b$, $c$ and $d$ with the same cubical index 200 constitute a cluster. All clusters are connected in the order of their cubical indices. The cluster with cubical index equals 200 has a preceding cluster with cubical index equals 50 and a succeeding cluster with cubical index equals 500. Node $d$ is the primary node of the nodes in its cluster.

Each node maintains a routing table recording the links to seven neighbors. The links include the node's predecessor and successor in its cluster, two primary nodes in the preceding and the succeeding cycles, one cubical neighbor, and two cyclic neighbors. We do not explain the cubical neighbor and the cyclic neighbors here since they are not closely related to the design of LORM. In Cycloid, an object is assigned to a node whose ID is the closest to the object's ID. If the target node of an object's ID $(k, a_{d-1}\ldots a_1a_0)$ is a participant, the object will be mapped to this node. Otherwise, the object 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 $k$. For example, in Figure 1, an object with ID=$(3, 200)$ is assigned to node $a = (3, 200)$. An object with ID=$(6, 200)$ is assigned to node $b = (5, 200)$ since there is no node with ID=$(6, 200)$ and $(5, 200)$ is the node ID that is the closest to $(6, 200)$.

The consistent hashing produces a bound of $O(\log n)$ imbalance of keys between nodes, where $n$ is the number of nodes in the system. Cycloid exhibits a more balanced distribution of key loads between the nodes than other DHTs on average. The balanced key load distribution helps LORM to prevent resource information imbalance, which is a severe problem in most grids centrally or hierarchically administered for resource discovery. For more information about Cycloid, please refer to [41]. Figure 2 shows a high level architecture for the application of Cycloid to grids for resource discovery. LORM relies on a single Cycloid with constant maintenance overhead. The goal of LORM is to address multi-attribute and range query resource discovery with low overhead and high efficiency.

## 3.2 LORM Framework and Algorithms

### *3.2.1 Resource Information Representation*

Usually, the resources required by applications are described by specifying a set of attributes such as available CPU time, memory, network bandwidth. It is a challenge to effectively locate resources across widely dispersed domains based on a list of predefined attributes. Without loss of generality, we assume that each resource is described by a set of attributes with globally known types and values/ranges or string description. E.g., “CPU=1000MHz”, “Free memory$\geqslant$2MB” or “OS=Linux”. We define *resource information* as information of available resources and resource queries. We use $\pi_a$ to denote the value/range (e.g., “2”) or string description (e.g., “Linux”) of a particular attribute $a$ (e.g., “Free memory” and “OS”). Resource information of a resource requester $j$ is represented in a set of 3-tuple representation: $< a, \pi_a, ip\_addr(j) >$, in which $ip\_addr(j)$ denotes the IP address of node $j$. The available resource information of node $i$ is represented in the form of $< a, \delta\pi_a, ip\_addr(i) >$, in which $\delta\pi_a$ is the $\pi_a$ of its available resource.

### *3.2.2 Resource Information Mapping*

Usually, the operation in resource discovery is to pool together the information of available resources in a number of *directory nodes*, and direct resource requests to these nodes, which return the resource owners of desired resources to the requesters. A directory node stores resource information in a directory. Recall that a Cycloid consists of a number of clusters, which together constitute a large cycle. As shown in Figure 3, the basic idea of LORM is to arrange each cluster to be responsible for the information of one attribute, and distributes the information among nodes within the cluster based on resource value/range or string description. Towards this end, LORM assigns each resource a Cycloid ID $(k, a_{d-1}a_{d-2}\dots a_0)$. Its cubical index $a_{d-1}a_{d-2}\dots a_0$ represents resource attribute $a$, and its cyclic index $k$ represents resource value $\pi_a$. Recall that in a Cycloid ID, the cubical indices $a_{d-1}a_{d-2}\dots a_0$ differentiate clusters, and the cyclic indices $k$ differentiate node positions in a cluster. As a result, the resource information is distributed in the Cycloid structure as shown in Figure 3.

LORM can build all nodes in a grid system into a Cycloid structure. For a grid system with $m$ types of resource attributes, if $m$ is smaller than the number of clusters in the Cycloid, some clusters will not be used. In this case, if $m$ is fixed, in order not to waste maintenance overhead, LORM can choose partial relatively stable nodes to build a Cycloid with $m$ clusters. If $m$ is larger than the number of clusters in the Cycloid, a cluster may be responsible for more than one resource attribute. In this case, a directory node in the cluster maintains one directory for each attribute.

In the following, we present the details of the design of LORM by answering three questions: (1) how to create resource ID? (2) how nodes report their resource information of available resources to their directory nodes? and (3) how nodes query resources from their directory nodes?

| Resource | Hash |
|---|---|
| CPU | 50 |
| Free memory | 200 |
| Disk | 500 |
| External memory | 800 |
| Software package | 1000 |
| Web service | 1200 |
| Bandwidth | 1800 |
| Database | 2047 |

TABLE 1
Resources and their Hash values.

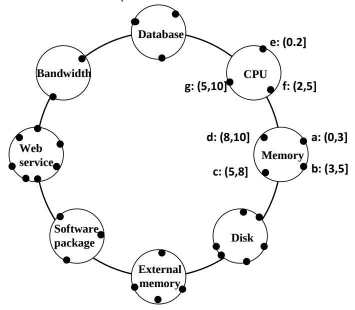


Fig. 3. Resource information distribution in LORM.

### *3.2.3 Resource ID*

Consistent hash functions such as SHA-1 are widely used in DHT networks for node or file ID due to its collision resistant nature. Using such a hash function, it is computationally infeasible to find a message that corresponds to a given hashed value, or find two different messages that produce the same message digest. The consistent hash function is effective in clustering messages based on message keywords such as resource attributes or attribute string descriptions, but cannot guarantee that an ordered list of numerical values will have a list of hash values in the same order. Therefore, consistent hash function is effective for cubical index based on resource attribute and for cyclic index based on attribute string description, whereas may not be appropriate for cyclic index based on resource values. Locality preserving hashing function [7] helps to solve the problem. We use $v$ to denote a value, use $H$ to denote a consistent hash function, and use $\mathcal{H}$ to denote a locality preserving hashing function.

**Definition 1 [7].** Hash function $\mathcal{H}$ is a locality preserving hashing function if it has the following property: $\mathcal{H}_{v_i} < \mathcal{H}_{v_j}$ iff $v_i < v_j$, and if an interval $[v_i, v_j]$ is split into $[v_i, v_k]$ and $[v_k, v_j]$, the corresponding interval $[\mathcal{H}_{v_i}, \mathcal{H}_{v_j}]$ must be split into $[\mathcal{H}_{v_i}, \mathcal{H}_{v_k}]$ and $[\mathcal{H}_{v_k}, \mathcal{H}_{v_j}]$.

We assume that each resource represented by a value has a minimum value and a maximum value denoted by $\pi_{min}$ and $\pi_{max}$, respectively. Since the ID length of a cluster is $d-1$, we define a locality preserving hashing function to be

$$\mathcal{H} = (\pi - \pi_{min}) \times (d-1)/(\pi_{max} - \pi_{min}),$$

where $\pi$ is a value for resource attribute $a$ in the range $[\pi_{min}, \pi_{max}]$. As a result, each resource value of attribute $a$ has a $\mathcal{H}$ value within $[0, d-1]$. Therefore, LORM assigns each piece of resource information a Cycloid ID denoted by *rescID*=($\mathbb{H}_{\pi_a}$,$H_a$), where

$$\mathbb{H}_{\pi_a} = \begin{cases} \mathcal{H}_{\pi_a} & \pi_a \text{ is a value} \\ H_{\pi_a} & \pi_a \text{ is a string.} \end{cases}$$

That is, the cubical index $H_a$ is the consistent hash function value of the resource attribute, and the cyclic index $\mathcal{H}_{\pi_a}$ is the locality preserving hashing value of resource value $\pi_a$ or the consistent hash function value $H_{\pi_a}$ of string description.

Thus, using the resource ID of a resource, a node can use the `Insert()` and `Lookup()` APIs to report and request resource information of the resource in a distributed manner.

### 3.2.4 Resource Information Reporting

Algorithm 1 shows the pseudocode for the different operations of a node in the LORM resource discovery. The operations include resource reporting and resource querying. In resource reporting, each node periodically reports its available resources with multiple attributes to the system via interface `Insert(rescID,rescInfo)` for each attribute, where *rescInfo*=$<a, \delta\pi_a, ip\_addr(i)>$ is the resource information (Lines 2-6 in Algorithm 1). Based on the Cycloid key assignment policy, in which a key is assigned to a node with the closest ID to its ID, the information of the same attribute will be mapped to the same cluster. Within each cluster, each node is responsible for the information of a resource whose cyclic index falls into the key space range it supervises. That is, the information of a resource is stored in a node whose cyclic index is the closest to the cyclic index in *rescID*. We call the node or the ID of the node the *root* of the key or *rescInfo*.

For example, for the resources and their hashed values listed in Table 1, the resource information will be stored in nodes in Figure 1 as illustrated in Figure 3. In the clusters responsible for memory and CPU, each node is responsible for the resource information in its range. For instance, node $a$ supervises range (0,3] and node $b$ supervises range (3,5] in their own cluster. The resource information of node $i$, $<mem, 2G, ip\_addr(i)>$, has *rescID*=($\mathcal{H}_2$,$H_{mem}$)=(2,200), and it will be routed to and stored in node $a$ via function `Insert(`$(\mathcal{H}_2, H_{mem}), <mem, \delta\pi_{mem}, ip\_addr(i)>$`)`. Similarly, if node $j$ has resource 1.8GHz CPU, the resource information will be reported and stored in node $e$ by `Insert(`$(\mathcal{H}_{1.8}, H_{CPU}), <CPU, \delta\pi_{CPU}, ip\_addr(j)>$`)`.

*Proposition 3.1:* In LORM, given a range query $[\pi_1, \pi_2]$ for a resource where $\pi_{min} \leqslant \pi_1 \leqslant \pi_2 \leqslant \pi_{max}$, a node that contains attribute value $\pi$ within $[\pi_1, \pi_2]$ must have an ID that satisfies root($\mathcal{H}_{\pi_2}$,$H_a$) $\geqslant$ ID $\geqslant$ root($\mathcal{H}_{\pi_1}$,$H_a$).

*Proof:* In LORM with $n$=$d{\cdot}2^d$ nodes, a node reports its resource information using the Cycloid interface `Insert(`$(\mathcal{H}_{\delta\pi_a}, H_a)$`,rescInfo)`. Attribute $a$ with value $\delta\pi_a$ will be stored in root($\mathcal{H}_{\delta\pi_a}$,$H_a$) whose ID is the closest to ($\mathcal{H}_{\delta\pi_a}$,$H_a$). According to Definition 1, because $\pi_1 \leqslant \pi \leqslant \pi_2$, the resource information of value $v$ will be stored in node $i$ that satisfies $root(\mathcal{H}(\pi_1)) \leqslant i \leqslant root(\mathcal{H}(\pi_2))$. □

### 3.2.5 Resource Information Querying

To simplify the description of resource querying, we let $A_Q$ denote the set of attributes in a query $Q$. A multi-attribute query $Q$ is composed of a set of sub-queries on each attribute in $A_Q$. For each sub-query, a node sends out `Lookup(rescID,`$<a, \pi_a, ip\_addr(i)>$`)`, where $\pi_a$ is the value or string description of the requested resource (Lines 9-12 in Algorithm 1). The query is routed to the directory node for the desired resource. All sub-queries are processed in parallel. For example, when node $k$ needs a multiple-attribute resource, say 1.8GHz CPU and 2GB memory, it sends requests `Lookup(`$\mathcal{H}_{1.8}, H_{CPU}$`)` and `Lookup(`$\mathcal{H}_2, H_{mem}$`)`, which will be resolved in parallel. In Figure 3, the queries will arrive at node $a$ and node $e$, which reply to the requester node $k$ with the requested resource information $<mem, \delta\pi_{mem}, ip\_addr(i)>$ $where$ $\delta\pi_{mem} = 2$ and $<CPU, \delta\pi_{CPU}, ip\_addr(j)>$ $where$ $\delta\pi_{CPU} = 1.8$. After the requester node receives all responses, it then concatenates the results in a database-like "join" operation based on $ip\_addr$ (Lines 15-18 in Algorithm 1). The results are the nodes that have the resources queried by the requester. For range queries such as "Free memory⩾2GB" and "CPU⩾1.8GHz", according to Proposition 3.1, in addition to responding with satisfied resource information in their own directories, nodes $a$ and $e$ forward the resource queries to their immediate successors in their own clusters. The successors check their own directories, respond satisfied resource information to the requester, and forward the queries to their immediate successors in their own clusters. This process is repeated until a successor has no satisfied resource information. If the requested resource range is less than a specified value, then the request receivers forward queries to their predecessors. If a requester's query has both lower and upper bounds such as "1GB⩽memory⩽2GB", it can first choose the middle value in [lower bound, upper bound] for the destination of resource query. The root which receives the query will generate and forward two queries to its successor and predecessor, respectively. The queries will be forwarded in both directions. A query receiver stops forwarding the query if its resource information is outside of the bound range.

**Algorithm 1** Pseudo-code for the operations of node $i$ in the LORM resource discovery.

```
1: /*periodically report resource information of its available resources with attributes $A$={$a_1$, $a_2$ ... $a_m$}*/
2: for each $a \in \{a_1, a_2 ... a_m\}$ do
3:     //$H_a$ is the consistent hash values of attribute a. $\mathbb{H}_{\delta\pi_a}$=$\mathcal{H}_{\delta\pi_a}$ if $\pi_a$ is a value, and $\mathbb{H}_{\delta\pi_a}$=$H_{\delta\pi_a}$ if $\pi_a$ is a string
4:     rescID=($\mathbb{H}_{\delta\pi_a}$,$H_a$)
5:     Insert(rescID,$<a, \delta\pi_a, ip\_addr(i)>$)
6: end for
7:
8: /*request resources using a multi-attribute range-query, $Q$={($a_1, \pi_{a_1}$),($a_2, \pi_{a_2}$)...($a_m, \pi_{a_m}$)}, $A_Q$={$a_1$,$a_2$...$a_m$}*/
9: for each $a \in A_Q$ do
10:     rescID=($\mathbb{H}_{\pi_a}$,$H_a$)
11:     lookup(rescID,$<a, \pi_a, ip\_addr(i)>$)
12: end for
13: /*after receiving responses of resource request Q*/
14: //IP_$addr(a)$ is a list of ip_addr as the response for lookup(rescID,$<a, \pi_a, ip\_addr(i)>$)
15: IP_$addr(Q)$=IP_$addr(a_1)$∩IP_$addr(a_2)$∩...∩IP_$addr(a_m)$
16: for each $ip\_addr(j) \in$ IP_$addr(Q)$ do
17:     SendMsg($ip\_addr(j)$) //request resource from node j
18: end for
```

### 3.2.6 Comparative Discussion

The multi-DHT-based approach and single-DHT-based decentralized approach need to probe all nodes in the system for a resource in a certain range in the worst case. Unlike these approaches, LORM reduces the probing scope for subsequent range queries after the `Lookup()` operation from network size $n$ to cluster size $d$ because all information about one attribute for all resources is gathered in a single cluster rather than spreads all over the system. Reducing the system-wide probing scope to cluster-wide scope significantly reduces the resource searching cost. To deal with the load imbalance caused by the distribution of resource information, load balancing methods [42] can be adopted into the LORM.

*Proposition 3.2:* With high probability[1], in LORM with $n = d \cdot 2^d$ nodes, the number of nodes that must be contacted to report resource information or to find a desired resource of an $m$-attribute resource is $O(md)$.

*Proof:* In a Cycloid DHT, the number of nodes that must be contacted for an object allocation or object owner using interface `Insert()` is $O(d)$ w.h.p. As resource information is allocated to nodes based on object allocation policy, the information reporting for one resource attribute also takes $O(d)$ hops. Therefore, it takes $O(md)$ for the information reporting of an $m$-attribute resource. For a range query, $d$ nodes in one cluster are needed to probe in the worst case. Therefore, the total number of nodes contacted for searching an $m$-attribute resource is $m(O(d) + d) = O(md)$. □

The CAN DHT [36] has a $m$-torus topology based on a $m$-dimensional Cartesian coordinate space. Thus, it can support multi-attribute range queries in a natural manner by using each dimension to represent an attribute. A resource is then represented by an $m$-dimensional CAN ID, in which the value in a dimension represents the resource value/description of the corresponding resource. Thus, for $m$ resources in a grid, $m$-torus CAN should be built. In LORM, $d \cdot 2^d = n$, $2^d = m$, the average path length is $d$ and the size of routing table is 7. In an $m$-torus CAN, the average routing path length is

$$(m/4)(n^{1/m}) > m/4 = 2^d/4 > d \ (when\ d > 4), \quad (1)$$

and the routing table size is

$$2m = 2 \cdot 2^d = 2^{d+1} > 7 \ (when\ d \geq 2). \quad (2)$$

Therefore, in a large-scale system when $d > 4$, the average path length and routing table size (i.e., maintenance overhead) of the $m$-torus CAN are larger than those of LORM. Also, even though some nodes only possess or query for part of the $m$ resources, they still need to generate the resource ID and process the resource reporting and querying based on the $m$ dimensions, which brings about unnecessary cost.

## 3.3 Dynamism-Resilient LORM

An effective resource discovery algorithm should work for grid systems with dynamic node joins and departures. Cycloid has a self-organization mechanism to maintain its structure and stored objects, which helps LORM to handle dynamism. When a node joins in the system, in addition to reporting its resources using `Insert(rescID,rescInfo)`, it receives the resource information in its responsible ID region from its neighbors based on the Cycloid's key assignment policy. For example, if node $h$ with ID (2,150) joins the system in Figure 3, then the resource information in the range (2,3] is transferred from node $a$ to the newly-joined node $h$. When a node departs from the system, it transfers its resource information to its neighbors. For instance, if node $a$ leaves, it transfers its resource information to node $d$ or $b$ based on the ID closeness. That is, the information in the range (0,1] is transferred to node $d$, and the information in the range (1,3] is transferred to node $b$. If node $a$ is the only node in its cluster, because the cluster ID of Memory 200 is closer to the cluster ID of CPU 50 than the cluster ID of disk 500 as shown in Figure 1, it transfers its information to the CPU cluster.

1. An event happens with high probability (w.h.p.) when it occurs with probability $1 - O(n^{-1})$.

For node failures or departures without warning, LORM resorts to the periodical resource information reporting by which the lost resource information will be recovered in its new root. Specifically, when a node receives a resource request, if it cannot locate the requested resource, it assumes that the old root of the resource information failed, and waits for a period of time $T$ which is the resource information reporting period. Within $T$, the lost resource information will be reported to the node, and the request can be resolved. With this self-organization mechanism, instead of relying on specific nodes, resource information is always stored in a node responsible for the ID region where the information ID locates even in the dynamic situation, and the `Lookup(rescID)` requests will always be forwarded to the node that has the required resource information.

*Proposition 3.3:* In a dynamic environment, in LORM, node joins and warned-departures will generate little adverse effect on resource querying, and a query for a resource whose root failed or departed without warning can be resolved within $T$ that is the resource information reporting period.

*Proof:* In Cycloid, any node joining or leaving leads to file transfer in order to maintain the consistency of file locations based on file assignment policy. A resource query is always forwarded to the root of the resource, so the node joining and leaving will result in little adverse effect on resource querying. If a root cannot resolve a resource request due to node failure or departure without warning, the lost resource information will be reported to the root in the next $T$. □

As indicated previously, when the number of resource attributes in the system $m$ is less than the number of clusters $c = 2^d$ in Cycloid when using all nodes to build the Cycloid, LORM can only choose stable nodes to form Cycloid in order to achieve $m = c$. To make our analysis applicable to both cases, in the analysis, we assume all resource discovery methods use the existing $n$ nodes in a grid system to build DHT overlay(s) for resource discovery, and define

$$\alpha = \begin{cases} 1 & m \leq c \\ \lceil m/c \rceil & m > c. \end{cases}$$

*Proposition 3.4:* In LORM with $n = d \cdot 2^d$ nodes, with the assumption that each resource attribute has $k$ pieces of resource information and the information is uniformly distributed, w.h.p., when an $(n+1)^{st}$ node joins or leaves the network, if the node is the only node in its cluster, responsibility for $O(k)$ pieces of resource information changes hands (and only to or from the joining or leaving node). Otherwise $O(k/d)$ pieces of resource information change hands.

*Proof:* Since each resource attribute has $k$ pieces of resource information and the information is uniformly distributed, a cluster has $\alpha k$ pieces of resource information and each node has $O(\alpha k/d)$=$O(k/d)$ pieces of resource information when the cluster has $d$ nodes. If a joining or leaving node is the only node in its cluster, it needs to hand over $O(\alpha k)$= $O(k)$ pieces of resource information. Otherwise, $O(\alpha k/d)$= $O(k/d)$ pieces of information change hands. □

## 4 Comparative Study and Analysis

There are mainly three classes of approaches dependent on DHTs for resource discovery in grids: multiple-DHT-based, single-DHT-based centralized and single-DHT-based decentralized. We use Mercury [5], SWORD [32] and MAAN [7] as representatives of multiple-DHT-based, single-DHT-based centralized and single-DHT-based decentralized classes, respectively. We analyze LORM in comparison with these approaches. LORM maps resource attribute and value or string description to two levels of the compound graph based Cycloid DHT. Mercury uses multiple DHTs with one DHT responsible for each attribute and maps resource value to each DHT. SWORD maps resource information including both attribute and value in a flat DHT, and MAAN maps attribute and value separately to a flat DHT. To be comparable, we use Chord for attribute hubs in Mercury, and we replace Bamboo DHT with Chord in SWORD.

In Mercury, for higher efficiency of resource querying, a node within one of the hubs can hold the data record while the other hubs can hold a pointer to the node. This strategy can also be applied to other methods to improve the efficiency. To make the different methods comparable, we don't consider this strategy in the comparative study. We analyze their performance in terms of structure maintenance overhead, resource information maintenance overhead, and the efficiency of resource discovery. In the analysis, we use *attribute value* to represent the locality preserving hash value of both attribute value and attribute string description. We use *directory size* to represent the number of resource information pieces in a directory.

### 4.1 Maintenance Overhead

*Proposition 4.1:* In a grid system with $n$ nodes and $m$ resource attributes, LORM can reduce the structure maintenance overhead of multiple-DHT-based resource discovery methods (e.g. Mercury) by a factor of no less than $m$.

*Proof:* LORM is based on Cycloid, in which each node is responsible for maintaining $7 \leq \log(n)$ neighbors in a large-scale system. In multiple-DHT-based methods such as Mercury, each node is responsible for maintaining $\log(n)$ neighbors for each DHT of one resource. Therefore, each node has $m\log(n)$ neighbors. The structure maintenance overhead can be saved by a factor of $\frac{m\log(n)}{7} \geq \frac{m\log(n)}{\log(n)} = m$. □

*Proposition 4.2:* In a grid system, the total number of resource information pieces in single-DHT-based decentralized resource discovery methods (e.g. MAAN) is twice of those in LORM, single-DHT-based centralized methods (e.g. SWORD) and multi-DHT-based methods (e.g. Mercury).

*Proof:* For each piece of resource information, MAAN splits its $a$ and $\pi_a$, and stores the two pieces of information separately, while LORM, single-DHT-based centralized (e.g. SWORD) and multi-DHT-based methods (e.g. Mercury) only store one information piece. Therefore, the size of the total resource information of MAAN is twice of others. Consequently, other methods generate half of the overhead for the maintenance of resource information in MAAN. □

*Proposition 4.3:* In a grid system with $n$ nodes and $m$ resource attributes, with the assumption that each type of resource attribute has $k$ pieces of resource information and its values are uniformly distributed, LORM can reduce the number of resource information pieces in a directory node in the single-DHT-based decentralized resource discovery methods (e.g. MAAN) by a factor of $\frac{d}{\alpha}(1+\frac{m}{n})$.

*Proof:* For $k$ pieces of resource information of a resource attribute, MAAN splits the attribute and value. $k$ pieces are stored in the same node, and the other $k$ pieces are uniformly distributed among the $n$ nodes based on the value. A directory node has a total of $k + m \cdot \frac{k}{n}$ pieces. LORM does not split the information, and all resource information of a particular resource attribute is in a cluster with $d$ nodes. With the uniform distribution assumption, each directory node is responsible for at most $\alpha k/d$ pieces of resource information. Therefore, LORM can reduce the total size of resource information in a directory node in MAAN by a factor of $\frac{k+m\cdot\frac{k}{n}}{\alpha k/d} = \frac{d}{\alpha}(1+\frac{m}{n})$. □

*Proposition 4.4:* In a grid system with $n$ nodes and $m$ resource attributes, with the assumption that each type of resource attribute has $k$ pieces of resource information and its values are uniformly distributed, LORM can reduce the resource information size in a directory node in single-DHT-based centralized methods (e.g. SWORD) by a factor of $\frac{d}{\alpha}$.

*Proof:* In LORM, all resource information of a particular resource attribute is in a cluster with $d$ nodes. With the uniform distribution assumption, each directory node has at most $\alpha k/d$ pieces of resource information. In SWORD, all resource information of a particular resource attribute is in a single node. Thus, LORM can reduce the resource information size in a directory node in SWORD by a factor of $\frac{k}{\alpha k/d} = \frac{d}{\alpha}$. □

*Proposition 4.5:* For any set of $n$ nodes and $m$ resource attributes, with the assumption that each type of resource attribute has $k$ pieces of resource information and its values are uniformly distributed, multi-DHT-based methods (e.g. Mercury) can achieve more balanced resource information distribution than LORM by a factor of $\frac{\alpha n}{dm}$.

*Proof:* The proof of Proposition 4.4 shows that each directory node is responsible for at most $\alpha k/d$ pieces of resource information in LORM. In Mercury, for one attribute, a node is responsible for $\frac{k}{n}$ pieces of resource information. Given $m$ resource attributes, each node is responsible for $\frac{mk}{n}$ pieces of resource information. Thus, Mercury can achieve more balanced resource information distribution than LORM by a factor of $\frac{\alpha k/d}{\frac{mk}{n}} = \frac{\alpha n}{dm}$. □

*Proposition 4.6:* Multi-DHT-based methods (e.g. Mercury) and LORM (when $m \leq c$) achieve more balanced resource information distribution than the single-DHT-based decentralized resource discovery methods (e.g. MAAN) and single-DHT-based centralized methods (e.g. SWORD).

*Proof:* Propositions 4.3 and 4.4 show that LORM achieves more balanced resource information distribution than the single-DHT-based decentralized method (e.g. MAAN) and single-DHT-based centralized method (e.g. SWORD) when $m \leq c$. Proposition 4.5 shows that multi-DHT-based methods (e.g. Mercury) achieves more balanced distribution than LORM. Therefore, multi-DHT-based methods and LORM achieve more balanced resource information distribution than the single-DHT-based decentralized methods and single-DHT-based centralized methods. □

## 4.2 Efficiency of Resource Discovery

We define the *contacted nodes* as the nodes that are involved in resource discovery, which include query routing nodes and directory nodes receiving resource queries.

*Proposition 4.7:* To discover resources for an $m$-attribute non-range resource query in an $n$-node network, w.h.p., LORM can reduce the total number of contacted nodes of single-DHT-based decentralized resource discovery methods (e.g. MAAN) by a factor of $\frac{\log n}{d}$.

*Proof:* For each piece of resource information including an attribute and value or string description, MAAN splits the attribute and value or string description and stores the information separately, while LORM only stores one piece of resource information. For each non-range resource query, LORM needs one DHT lookup, while MAAN needs two DHT lookups for each attribute: attribute name and value. For an $m$-attribute resource query, LORM needs $m$ DHT lookups, and MAAN needs $2m$ DHT lookups. Hence, for one resource query, the number of lookups in MAAN is $\frac{2m}{m}$=2 times of LORM. On the average case, one lookup needs $\log n/2$ hops in Chord [46] and $d$ hops in Cycloid [41]. Thus, LORM can reduce the total number of contacted nodes of MAAN by a factor of $\frac{2\log n/2}{d} = \frac{\log n}{d}$. □

*Proposition 4.8:* To discover resources for an $m$-attribute non-range resource query in an $n$-node network, w.h.p., multi-DHT-based methods (e.g. Mercury) and single-DHT-based centralized methods (e.g. SWORD) can reduce the total number of contacted nodes of single-DHT-based decentralized resource discovery methods (e.g. MAAN) by a factor of 2.

*Proof:* For each non-range resource query, the multi-DHT-based methods (e.g. Mercury) and single-DHT-based centralized methods (e.g. SWORD) need one DHT lookup, while MAAN needs two DHT lookups for each attribute; attribute name and value. For an $m$-attribute resource query, the former methods need $m$ DHT lookups, and MAAN needs $2m$ DHT lookups. Hence, the former methods can reduce the number of contacted nodes in MAAN by a factor of 2. □

When a range resource query is routed to its root, the root node checks its directory for the range query. Then, in SWORD, the resource searching stops; in Mercury and MAAN, the node forwards the query to its successor or predecessor according to their closeness to the queried range; in LORM, the node forwards the query to its successor or predecessor in its cluster according to their closeness to the queried range. The nodes receiving the query repeat the process. We call the nodes that receive a resource query and check their directories for the queried resource *visited nodes* of the resource query. Visited nodes are included in the contacted nodes.

*Proposition 4.9:* In an $n$-node network, w.h.p., LORM can reduce at least $\frac{m(n-d)}{4}$ visited nodes to discover required resource for an $m$-attribute range resource query in system-wide range resource discovery methods (e.g. MAAN and Mercury), and SWORD can reduce $\frac{md}{4}$ visited nodes in LORM, in the average case.

*Proof:* In Mercury, a range query needs $\frac{n}{2}$ contacted nodes in the worst case. Thus, on the average case, the total number of visited nodes for an $m$-attribute resource range query on the average case is $m(1+\frac{n}{4})$. MAAN has two lookups for each attribute query. $m$-attribute resource query needs $m(2+\frac{n}{4})$ hops. In LORM, the nodes needed to be visited for a resource query is $m(1+\frac{d}{4})$ on the average case. Therefore, LORM can reduce at least $m(1+\frac{n}{4}) - m(1+\frac{d}{4}) = \frac{m(n-d)}{4}$ visited nodes for an $m$-attribute resource query in system-wide range resource discovery methods such as MAAN and Mercury, on the average case. Because SWORD doesn't need to forward query for range query, it reduces $m(1+\frac{d}{4}) - m = \frac{md}{4}$ contacted nodes in LORM. □

*Proposition 4.10:* In an $n$-node network, w.h.p., LORM can reduce at least $mn$ contacted nodes to discover the required resource for an $m$-attribute resource range query in system-wide range resource discovery methods (e.g. MAAN and Mercury), in the worst case.

*Proof:* Mercury uses one DHT for each attribute. For each attribute in a resource requester, it needs $\log n$ hops for the request to reach its root node [46]. After that, $n$ nodes need to be probed for the range query, in the worst case, because all resource information of the attribute spreads over the $n$ nodes. Therefore, for each attribute query, $\log n + n$ hops are needed in the worst case. For a resource query with $m$ attributes, Mercury needs to contact $m(\log n{+}n)$ nodes. MAAN has two lookups for each attribute query since it lookups attribute name and value or string description separately. Only one lookup needs system-wide probing on $n$ nodes. Consequently, each query of an attribute needs $(2\log n{+}n)$ hops, and $m$-attribute resource query needs $m(2\log n + n)> m(\log n + n)$ hops in MAAN. The number of nodes that need to be contacted in LORM for a resource query is $m \cdot d \leq m \cdot \log n$ in the worst case. Therefore, LORM can reduce at least $m(\log n + n) - m \cdot \log n = mn$ contacted nodes for an $m$-attribute resource query in system-wide range resource discovery methods such as MAAN and Mercury, in the worse case. □

## 5 Performance Comparison

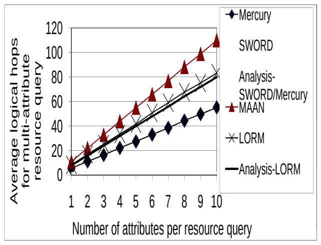
This section presents the performance evaluation of LORM in average case in comparison with Mercury [5], SWORD [32], MAAN [7]. We designed and implemented a simulator in Java for the evaluation of LORM, Mercury, Sword and MAAN. To be comparable, we used Chord for Mercury and SWORD. In the experiments, the dimension was set to 8 in Cycloid, and 11 in Chord, and each DHT had 2048 nodes. We assumed there were $m = 200$ resource attributes, and each attribute had $k = 500$ values. The size distribution of process CPU and memory requirements, and process lifetime fit a bounded Pareto distribution [55, 35, 50, 21]. The bounded Pareto distribution was also used for node capacity [28]. Therefore, we used Bounded Pareto distribution function to generate resource values owned by a node and requested by a node. This distribution reflects real world where there are machines with capacities that vary by different orders of magnitude. The resource attributes in a node resource request were randomly generated.

## 5.1 Maintenance Overhead

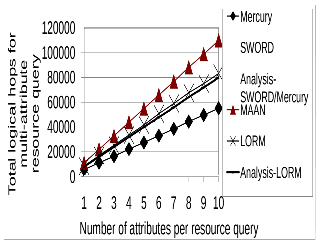
In DHT overlays, each node needs to maintain a number of neighbors (outlinks) in its routing table. Therefore, the maintenance of routing tables or the outlinks constitutes a large part of the DHT overlay maintenance overhead. Proposition 4.1 shows that LORM can reduce the DHT maintenance overhead of Mercury by a factor of no less than $m$. Thus, we use “Analysis$-$LORM” to represent the experiment results of Mercury divided by $m = 200$. Figure 4(a) plots the number of outlinks maintained by each node in Mercury, “Analysis$-$LORM” and LORM versus network size. From the figure, we can see that the number of outlinks per node in LORM is less than that of “Analysis$-$LORM”. Mercury has dramatically more outlinks per node than LORM. The experiment results are consistent with Proposition 4.1. Recall that Mercury has multiple DHTs with each DHT responsible for one resource attribute, such that each node has a total number of outlinks equals to the product of routing table size and the number of DHTs. It means that in Mercury, each node needs much higher overhead to maintain its outlinks than in LORM.

In addition to the outlinks, a directory node also needs to maintain resource information in its directory. It is desirable to distribute the information among nodes uniformly so that the information maintenance overhead as well as the resource discovery load can be distributed among nodes to avoid bottlenecks. The directory size is a metric for the balance of the resource information distribution. The average directory size is the total number of resource information pieces divided by the total number of nodes.

Proposition 4.2 proved that LORM can reduce the total (average) directory size of MAAN by half. Proposition 4.3 proved that LORM can reduce the directory size of MAAN by $d(1+\frac{m}{n})$. Figure 4(b) plots the experiment results of the average and the 1st and 99th percentiles of directory size per node in MAAN and LORM. It also plots the analysis results of LORM based on MAAN according to Propositions 4.2 and 4.3. That is, in the figure, the analysis results of the 1st and 99th percentiles are calculated as the experiment results of MAAN divided by the factor of $d(1+\frac{m}{n}) = 8 \times (1+\frac{200}{2048}) = 8.78$, and the analysis results of the average directory size are calculated as the experiment results of MAAN divided by 2. We can see that the experiment results of the average directory



(a) Average number of contacted nodes (Propositions 4.7 and 4.8)



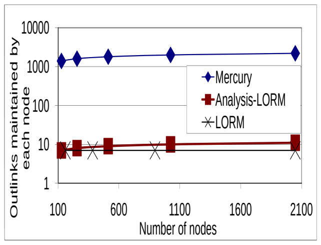
(b) Total number of contacted nodes (Propositions 4.7 and 4.8)

Fig. 5. The number of hops for query routing in different resource discovery approaches.

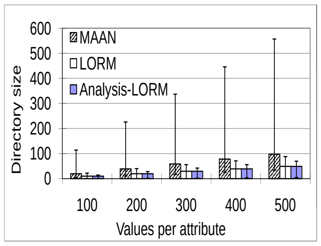
size of LORM match the analytical results in Proposition 4.2. Recall that MAAN separates resource attribute and value or string description of a piece of resource information, and stores the information separately. Therefore, MAAN doubles the total resource information size and needs information maintenance overhead twice as high as others. The experiment results of the 1st and 99th percentiles are close to the analytical results. The experimental results of the 99th percentile of LORM are slightly higher than the analytical results. This is because that the resource values are randomly chosen in the experiment and are not completely uniformly distributed. In addition, all resource information of the same attribute is collected in one cluster with only d=8 nodes. It is very likely that a node has much more resource information than other nodes in the same cluster, resulting in slightly higher 99th percentile than that when the values are uniformly distributed.

Proposition 4.4 proved that LORM can reduce the directory size of SWORD by a factor of $d$. Figure 4(c) plots the experiment results of the average and the 1st and 99th percentiles of directory size per node in SWORD and LORM. It also plots the analytical results of LORM based on the experiment results of SWORD. The analytical results of the 1st and 99th percentiles are the experiment results of SWORD divided by the factor of $d$. Proposition 4.2 implies that the average directory size of LORM equals to SWORD. Therefore, the analytical results of the average directory size of LORM in the figure equal to the experimental results of SWORD. We can see that the experimental results of the average directory size of LORM match the analytical results as implied in Proposition 4.2. The experimental results of the 99th percentile are only slightly higher than the analytical results in Proposition 4.4. This is because that the attribute values are randomly distributed among $d$ nodes, and some nodes have more resource information than others with random distribution.

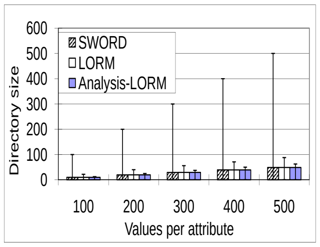
Figure 4(d) plots the experimental results of the average and the 1st and 99th percentiles of directory size per node in Mercury and LORM. It also plots the analytical results of LORM based on Mercury. Proposition 4.5 proved that Mercury can achieve more balanced resource information distribution than LORM by a factor of $\frac{n}{dm}$. Therefore, the analytical results of the 1st and 99th percentiles of LORM in the figure are calculated as the experimental results of Mercury multiplied and divided by the factor of $\frac{n}{dm} = \frac{2048}{8\times 200} = 1.28$, respectively. Proposition 4.2 implies that the average directory size of LORM equals that of Mercury. Thus, the analytical results of



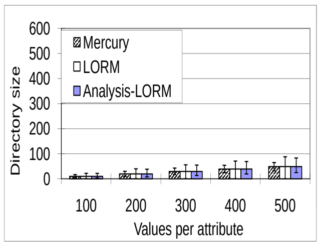
(a) Average number of outlinks (Proposition 4.1)



(b) MAAN versus LORM (Proposition 4.2 & Proposition 4.3)

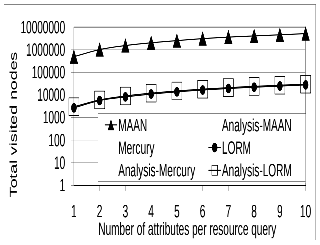


(c) SWORD versus LORM (Proposition 4.4)

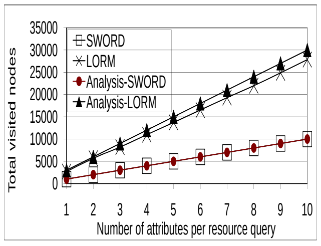


(d) Mercury versus LORM (Proposition 4.5)

Fig. 4. Overhead in different resource discovery approaches.



(a) Visited nodes of MAAN/Mercury and LORM (Proposition 4.9)



(b) Visited nodes of SWORD and LORM (Proposition 4.9)

Fig. 6. Searching cost in different resource discovery approaches.

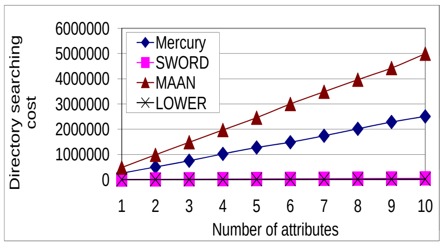
average directory size of LORM in the figure are calculated as the experimental results of Mercury. We can see that the experimental results of the average directory size of LORM match the analytical results as implied in Proposition 4.2. The experimental results of the 99th percentile are also only slightly higher than the analytical results due to the same reason observed in Figure 4(b) and (c). In addition, the results of the 1st percentile are lower than the analytical results in Proposition 4.4. This is because when attribute values are randomly selected, some values may not be chosen and hence some nodes in a cluster in LORM may not be assigned resource information.

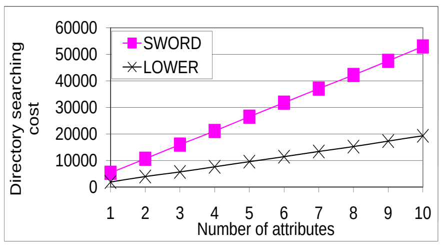
In general, the experimental results of LORM match the analytical results. From Figures 4(b), (c) and (d), we can observe that MAAN and SWORD exhibit significantly larger variance of directory size than Mercury and LORM. MAAN and SWORD distribute resource information to directory nodes based on resource attribute. As there are 200 resource attributes, the information is accumulated in 200 nodes among 2048 nodes, leading to large variance of directory size. On the other hand, Mercury uses one DHT for each attribute, and classifies resource information based on value in each DHT. The widespread information distribution helps to distribute resource information uniformly. LORM arranges different Cycloid clusters to be responsible for resource information based on resource attribute and allocates information to a node based on its range, leading to more balanced information distribution. Therefore, Mercury and LORM can achieve more balanced distribution of load due to resource information maintenance and resource discovery operation. This result is in agreement with Proposition 4.6.

## 5.2 Efficiency of Resource Discovery

For a non-range query, Proposition 4.7 shows that LORM can reduce the total number of contacted nodes of MAAN by a factor of $\frac{\log n}{d} = \frac{11}{8}$; Proposition 4.8 shows that Mercury and SWORD can reduce the total number of contacted nodes of MAAN by a factor of 2. We conducted an experiment to evaluate the efficiency of different resource discovery approaches. We varied the number of attributes in a query from 1 to 10 with step size of 1. The logical hop metric is measured by the number of hops traversed during a search until a query for resource information reaches its root (i.e., contacted nodes). We randomly chose 100 nodes and let each node send 10 resource queries. Figure 5(a) and (b) show the experimental results of the average and total logical hops for multi-attribute resource queries versus the number of attributes in a resource query in Mercury, SWORD, MAAN and LORM. The figures also plot the analytical results of LORM and SWORD/Mercury based on the experimental results of MAAN denoted by "Analysis-LORM" and "Analysis-SWORD/Mercury". That is, "Analysis-LORM" and "Analysis-SWORD/Mercury" are the experimental results of MAAN divided by $\frac{11}{8}$ and 2, respectively. Because the difference between Mercury, SWORD and "Analysis-SWORD/Mercury" is no more than 0.3 in Figure 5(a), and their difference is no more than 800 in Figure 5(b), these curves are completely overlapped. In order to make the figure clear, we only draw the curve of Mercury and also use it to represent SWORD and "Analysis-SWORD/Mercury". From the figure, we can see that the experimental results of LORM is very close to the analytical results, and the experimental results of SWORD/Mercury exactly match the the analytical results in Proposition 4.7 and 4.8.

Comparing the different methods, we can see that MAAN generates the highest number of contacted hops and SWORD/Mercury produce the least number of contacted hops. This is because MAAN needs two lookups for resource attribute and value, and others only need one lookup. The reason that LORM has higher number of hops than Mercury and SWORD is due to their time complexity of lookups. Chord has a time complexity of $O(\log n)$ per query, and Cycloid has a time complexity of $O(d)$ per query due to its constant-degree feature. Specifically, LORM is based on Cycloid which has a constant 7 neighbors per node, whereas Mercury and SWORD have $\log n = 12$ neighbors each node. Due to the tradeoff between the number of neighbors per node and lookup path length, Cycloid has high lookup path length. These results are consistent with the file lookup path length in [41]. The


(a) Directory searching cost


(b) Directory searching cost of LORM and SWORD

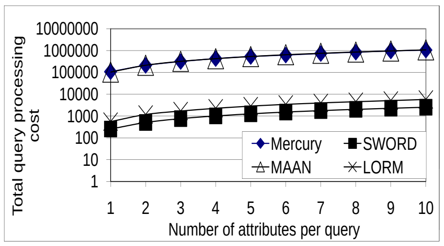
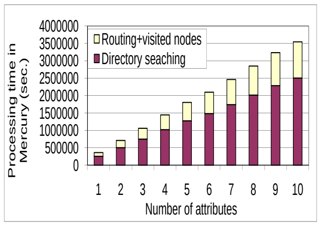
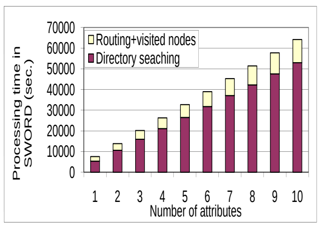
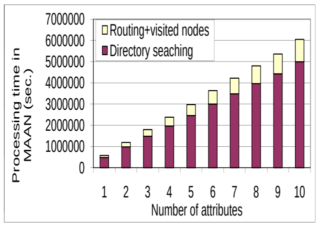

(c) Total processing cost

Fig. 7. Searching cost in different resource discovery approaches.


(a) Mercury


(b) SWORD


(c) MAAN

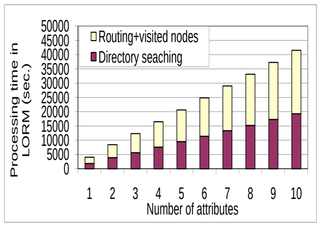
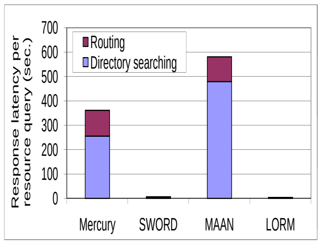

(d) LORM

Fig. 8. Searching latency in different resource discovery approaches.

figures also show that the average number and total number of logical hops increase as the number of attributes in each resource request grows. It is because a node needs to send out multiple queries for multiple attributes. This result is in agreement with the analytical result in Proposition 3.4 that the number of contacted nodes increases in proportion with the number of attributes in a resource query.

For a range resource query, after the root node for the resource information is reached, the root node needs to probe other nodes for the requested resources in a range. We use the number of visited nodes to represent resource searching efficiency. More visited nodes imply less efficiency in resource searching. The proof in Proposition 4.9 shows that to query an $m$-attribute resource with range requirement in an $n$-node network, the total number of visited nodes is $m(1+\frac{n}{4})$ in Mercury, $m(2+\frac{n}{4})$ in MAAN, $m(1+\frac{d}{4})$ in LORM and $m$ in SWORD. Based on the analysis, we calculated the number of visited nodes for one query. It is $m(1+\frac{n}{4})=513m$ in Mercury, $m(2+\frac{n}{4})=514m$ in MAAN, $m(1+\frac{d}{4})=3m$ in LORM, and $m$ in SWORD. The total number of visited nodes for 1000 queries is the product of the result and 1000. Figure 6(a) and Figure 6(b) plot the experiment results and analytical results of the number of visited nodes versus the number of attributes per query. Figure 6(a) demonstrates the results of MAAN/Mercury and LORM, and Figure 6(b) shows the results of SWORD and LORM. In Figure 6(a), the results of MAAN, Mercury, "Analysis-MAAN" and "Analysis-Mercury" are completely overlapped because their values differ no more than 70000. Therefore, for clarity, we only show the results of MAAN to represent itself and the others. In the figure, the y axis is shown in logarithmic scale and MAAN and Mercury have dramatically large results. Also, the experimental results and analytical results of LORM are completely overlapped. Actually, Figure 6(b) shows that the experimental results of LORM are a little lower than its analytical results.

Except that the experiment results of LORM are marginally lower than its analysis results, The experimental results of other methods are almost consistent with their analytical


(a) Processing time

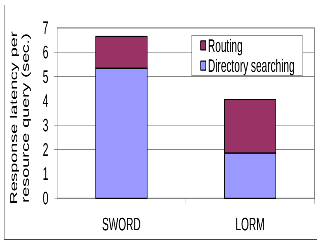

(b) Processing time of SWORD & LORM

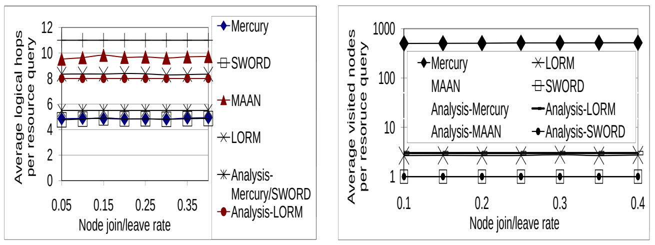
Fig. 9. Efficiency of different resource discovery approaches.

results. Mercury and MAAN visit tremendously more nodes than SWORD and LORM. Recall that Mercury and MAAN accumulate resource information based on attribute value, which spreads along the entire DHT ID space. They need to probe nodes along a very large ID space. In contrast, SWORD accumulates resource information based on node attribute. All information of a particular attribute is in one directory node, and no node needs to be probed. Therefore, a resource query for an $m$-attribute resource needs $m$ visit nodes. SWORD's experimental results exactly match its analytical results. LORM stores resource information of a specific attribute name in a cluster, and only the nodes in the cluster should be probed. This limits the node probing scope to a cluster rather than the entire system. As a result, SWORD and LORM incur much less cost for a range query than Mercury and MAAN.

We note SWORD achieves efficiency at the cost of high information maintenance overhead in directory nodes, and high directory searching cost. To verify this, we record the directory size of each visited node, and use the sum of the sizes to represent the directory searching cost as shown in Figure 7(a). Since the results of LORM and SWORD are overlapped, we show their results in Figure 7(b) for clear demonstration. We can see that SWORD leads to higher directory searching cost than LORM due to its large directory. Although LORM needs to probe nodes in a cluster, because of balanced information distribution, it still has a lower directory searching cost than SWORD. From Figure 7(a), we can also find that MAAN and Mercury generate significantly higher cost, due to their large number of probed nodes. Also, MAAN produces higher cost than Mercury because of its doubled directory size. Considering that the number of visited nodes is significantly larger than the routing hops, we can conclude that LORM is the most efficient in terms of the total searching cost for resource queries. Since a node communication costs more than a directory entry search, we assume that each node contact consumes 0.2 unit cost, while each directory entry visit consumes 0.01 unit cost. Figure 7(c) plots the total resource query processing cost. We can find that the results are almost the same as the results in Figures 6(a) and (b) because the latency of probing constitutes the main part of the latency.

Cost aside, resource querying latency is another important metric to measure the efficiency of a resource discovery approach. Since node communication takes longer time than directory searching at a node and directory searching latency is based on the directory size, we assume that the time for each node communication was 0.2 second, and the time for each directory searching is the product of 0.01 second and the directory size. Such assumption does not affect the relative performance comparison between different approaches. Communication nodes include the nodes along the route for forwarding resource query and the visited nodes. Figure 8 shows the breakdown of total resource querying latency into the factors of node communication and directory searching. In Mercury, SWORD and MAAN, directory searching constitutes the major part. Mercury and MAAN need to probe nodes for range query in the entire system, leading to high directory searching delay. Though SWORD does not need to probe nodes, it has all the information of one resource attribute in one node. Therefore, all queries for the same resource attribute will reach a node which will become a bottleneck, resulting in high searching delay. With limited probing scope within a cluster and more balanced resource information distribution, LORM has much less directory searching delay. Although LORM has longer path length for each lookup operation, its constraint probing scope saves the latency for node communication.

To be comparable, we choose the results in the case of one-attribute resource query and put them in one figure. Figure 9(a) and (b) show the average response latency for all resource queries. We can observe that Mercury and MAAN take dramatically longer response latency than SWORD and LORM, and the latency of MAAN is longer than Mercury. It is shown that Mercury and MAAN take approximately the same time for routing, and MAAN has longer directory searching latency than Mercury. The reason that they have almost the same routing latency is because all queries for all attributes are processed at the same time. However, MAAN produces larger directory size due to its doubled resource information, leading to longer latency for directory searching. From Figure 9(b), we can see that LORM has a shorter response latency than SWORD, though it takes longer latency for routing. It is because Cycloid has a little longer query routing path length than Chord, and LORM needs to probe nodes in a cluster for range queries while SWORD does not need to probe nodes. However, LORM introduces more



(a) Average number of contacted nodes (Propositions 3.3, 4.7, and 4.8) (b) The number of visited nodes (Propositions 3.3, and 4.9)

Fig. 10. Efficiency of different resource discovery approaches in churn.

balanced resource information distribution, while SWORD accumulates information of a specific attribute to a single node, so LORM has less directory searching latency than SWORD. The results show that LORM is most efficient in resource query processing.

## 5.3 Performance in a Dynamic Environment

This section evaluates the efficiency of the LORM in a highly dynamic environment in comparison with other approaches with the DHT stabilization mechanism. In this experiment, the resource join/departure rate $R$ was modelled as a Poisson process as in [46]. For example, there is one resource join and one resource departure every 2.5 seconds with $R = 0.4$. We varied $R$ from 0.1 to 0.5, with step size of 0.1. We set the number of total resource requests to 10000. Experiment results show that there were no failures in all test cases. Figure 10(a) shows the experimental and analytical results of the average number of logical hops for a non-range resource query as $R$ changes. The analytical results are drawn from the proof of Proposition 4.7 and 4.8. Compared with the logical hop evaluation in Figure 5 in a static environment, we can see that the measured number of hops in the presence of node joining and leaving is very close to that in a static environment and does not change with the rate $R$. This is consistent with Proposition 3.3 that node joins and departures generate little adverse effect on resource querying. We can also observe that the analytical results are slightly higher than the experiment results. This is because that the lookup path is reduced sometimes due to the node departures. The results are also consistent with Proposition 4.7 and 4.8 in terms of the number of contacted hops for a resource query.

Figure 10(b) shows the analytical results and experimental results of the average visited nodes per range resource query. The results of Mercury, MAAN, “Analysis-Mercury” and “Analysis-Mercury” are overlapped since their results differ no more than 30. Thus, we only draw the result of Mercury. The analytical results are drawn from the proof of Proposition 4.9. First, we can see that the experimental results are consistent with the analytical results. Because Mercury and MAAN are system-wide range resource discovery methods that distribute resource information in all nodes based on attribute values, they incur much more node communication for a resource query due to system-wide probing. Second, the results are consistent with the results in Figure 6 in the static situation. Thus, the results are in agreement with Proposition 3.3 that dynamism generates little adverse effect on the efficiency of resource querying. In conclusion, the experimental results confirm that LORM can effectively resolve resource queries in a dynamic environment.

### 5.4 Summary

In summary, LORM leads to superior performance over existing multi-attribute range querying methods:

- It generates lower structure maintenance overhead.
- It produces more balanced load distribution among directory nodes for directory information maintenance and resource discovery.
- It is more efficient in querying with respects of the number of hops traversed in routing and the cost of searching the directory.
- It is more efficient in query response latency.
- It is churn resilient.

## 6 Conclusions

Resource discovery is a critical issue for grid systems in which applications are composed of hardware and software resources. Previous resource discovery approaches either depend on multiple DHTs with each DHT responsible for a resource or rely on one DHT by pooling resource information of an attribute in a single node, leading to high maintenance overhead or inefficiency due to load imbalance. In this paper, we propose a Low-Overhead Range-query Multi-resource discovery approach (LORM). Unlike most previous resource discovery methods which depend on multiple DHTs with each DHT responsible for a resource, LORM relies on a single DHT with constant maintenance overhead to achieve range-query multi-attribute resource discovery with low overhead. It avoids bottlenecks by achieving a balanced distribution of load due to resource information maintenance as well as resource discovery operation itself. Furthermore, LORM is able to deal with dynamic node changes and variation of resource availability. We experimentally and analytically study the performance of LORM in comparison with the previous resource discovery methods with regards to their structure maintenance overhead, resource information maintenance overhead and resource searching efficiency. We show the consistency of the analytical results with the experimental results. Both simulation and analytical results show the superiority of LORM in comparison with other representative approaches in terms of overhead cost and efficiency of range query and multi-attribute resource discovery. We plan to further explore and elaborate upon the LORM design to discover resources based on semantic information.

## Acknowledgements

This research was supported in part by U.S. NSF grants OCI-1064230, CNS-1049947, CNS-1025652, CNS-1025649, CNS-0917056, CNS-0702488, CNS-0914330, and CCF-1016966, Microsoft Research Faculty Fellowship 8300751, and Sandia National Laboratories grant 10002282.

## References

[1] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modelling with Nimrod/G: killer application for the global grid? In *Proc. of IPDPS*, 2000.

[2] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proc. of P2P*, 2002.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. of SODA*, 1994.

[4] J. L. Bentle, J. H. Friedman, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[5] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proc. of ACM SIGCOMM*, pages 353–366, 2004.

[6] D. Boukhelef and H. Kitagawa. Efficient management of multidimensional data in structured peer-to-peer overlays. In *Proc. of VLDB*, 2009.

[7] M. Cai, M. Frank, J. Chen, and P. Szekely. MAAN: A multi-attribute addressable network for grid information services. *Journal of Grid Computing*, 2004.

[8] M. Cai and K. Hwang. Distributed aggregation algorithms with load-balancing for scalable grid resource monitoring. In *Proc. of IPDPS*, 2007.

[9] S. Castelli, P. Costa, and G. P. Picco. Hypercbr: Large-scale content-based routing in a multidimensional space. In *Proc. of INFOCOM*, 2008.

[10] C. Chen and D. Agrawal. dBCube: a new class of hierarchical multiprocessor interconnection networks with area efficient layout. *TPDS*, 4(12):1332 – 1344, 1993.

[11] H. Chen, H. Jin, J. Wang, L. Chen, Y. Liu, , and L. M. Ni. Efficient multi-keyword search over p2p web. In *Proc. of WWW*, 2008.

[12] A. Datta, M. Hauswirth, R. John, R. Schmidt, and K. Aberer. Range queries in trie-structured overlays. In *Proc. of P2P*, 2005.

[13] C. Doulkeridis, A. Vlachou, K. Nrvg, Y. Kotidis, and M. Vazirgiannis. Multidimensional routing indices for efficient distributed query processing. In *Proc. of CIKM*, 2009.

[14] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: a computation management agent for multiinstitutional grids. In *Proc. HPDC*, 2001.

[15] A. Fu, P. M. S. Chan, Y. L.Cheung, and Y. S. Moon. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *VLDB Journal*, 9(2):154–173, 2000.

[16] P. Ganesan, B. Yang, and H. Garcia-molina. One torus to rule them all: Multi-dimensional queries in p2p systems. In *Proc. of WebDB*. ACM Press, 2004.

[17] J. Gao and P. Steenkiste. An adaptive protocol for efficient support of range queries in DHT-based systems. In *Proc. of ICNP*, 2004.

[18] D. Guo, H. Chen, Y. Hec, H. Jin, C. Chen, H. Chen, Z. Shu, and G. Huang. Kcube: A novel architecture for interconnection networks. *Information Processing Letters*, 2010.

[19] D. Guo, Y. Liu, and X. Li. Bake: A balanced kautz tree structure for peer-to-peer networks. In *Proc. of INFOCOM*, 2008.

[20] D. Guo, J. Wu, H. Chen, and X. Luo. Moore: An extendable peer-to-peer network based on incomplete kautz digraph with constant degree. In *Proc. of INFOCOM*, 2007.

[21] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *TOCS*, 1997.

[22] H. V. Jagadish. Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *Proc. of ICDE*, page 34, 2006.

[23] K. V. R. Kanth, D. Agrawal, A. E. Abbadi, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. of ACM SIGMOD*, 1998.

[24] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. 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.

[25] D. Li, X. Lu, and J. Wu. FISSIONE: a scalable constant degree and low congestion DHT scheme based on Kautz graphs. In *Proc. of INFOCOM*, 2005.

[26] J. Li and S. Vuong. Grid resource discovery based on semantic p2p communities. In *Proc. of ACM-SAC*, 2006.

[27] M. Li, W.-C. Lee, and A. Sivasubramaniam. Dptree: A balanced tree based indexing framework for peer-to-peer systems. In *Proc. of ICNP*, 2006.

[28] Q. Li, L. Feng, J. Pei, S. Wang, X. Zhou, and Q. Zhu. Advances in Data and Web ManagementJoint International Conferences. *Lecture Notes in Computer Science*, 2009.

[29] B. Liu, W.-C. Lee, and D. L. Lee. Supporting complex multi-dimensional queries in p2p systems. In *Proc. of ICDCS*, 2005.

[30] V. March and Y. M. Teo. Multi-attribute range queries in read-only DHT. In *Proc. of ICCCN*, pages 419–424, 2005.

[31] M. Marzolla, M. Mordacchini, and S. Orlando. Peer-to-peer systems for discovering resources in a dynamic grid. *Parallel Computing*, 2006.

[32] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable wide-area resource discovery. Technical Report TR CSD04-1334, EECS Dept., Univ. of California, Berkeley, 2004.

[33] D. Papadias, Y. Theodoridis, and E. Stefanakis. Multidimensional range query processing with spatial relations. *Geographical Systems*, 4:343–365, 1997.

[34] F. P. Preparata and J. Vuillemin. The cube-connected cycles: A versatile network for parallel computation. *CACM*, 1981.

[35] K. Psounisa, P. M. Fernandezb, B. Prabhakarc, and F. Papadopoulosd. Systems with multiple servers under heavy-tailed workloads. *Performance Evaluation*, 2005.

[36] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. of ACM SIGCOMM*, pages 329–350, 2001.

[37] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*, 2001.

[38] T. Schtt, F. Schintke, and A. Reinefeld. A structured overlay for multi-dimensional range queries. In *Proc. of Euro-Par*, 2007.

[39] H. Shen. A p2p-based intelligent resource discovery mechanism in internet-based distributed systems. *JPDC*, 2008.

[40] H. Shen, A. Apon, and C. Xu. LORM: Supporting low-overhead p2p-based range-query and multi-attribute resource management in grids. In *Proc. of ICPADS*, 2007.

[41] H. Shen, C. Xu, and G. Chen. Cycloid: A scalable constant-degree P2P overlay network. *Performance Evaluation*, 63(3):195–216, 2006.

[42] H. Shen and C.-Z. Xu. Hash-based proximity clustering for efficient load balancing in heterogeneous dht networks. *JPDC*, 2008.

[43] X. Shen and Z. Li. Multi-dimensional queries in dht-based peer-to-peer systems. In *Proc. of SKG*, 2009.

[44] Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Proc. of P2P*, 2005.

[45] D. Spence and T. Harris. XenoSearch: Distributed resource discovery in the XenoServer open platform. In *Proc. of HPDC*, pages 216–225, 2003.

[46] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM TON*, 11(1):17–32, 2003.

[47] H. Sun, J. Huais, Y. Liu, and R. Buyya. RCT: A distributed tree for supporting efficient range and multi-attribute queries in grid computing. *Future Generation Computer Systems*, 2008.

[48] X. Sun. SCAN: A small-world structured p2p overlay for multi-dimensional queries. In *Proc. of WWW*, 2007.

[49] L. Weng, U. Catalyurek, T. Kurc, G. Agrawal, and J. Saltz. Optimizing multiple queries on scientific datasets with partial replicas. In *Proc. of Grid Computing*, 2007.

[50] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *TON*, 1997.

[51] D. Wu, Y. Tian, and K.-W. Ng. Roogle: Supporting efficient high-dimensional range queries in P2P systems. In *Proc. of Euro-Par*, 2006.

[52] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *Proc. of ACM SIGCOMM*, 2004.

[53] L. Zhang, Z. Wang, and D. Feng. Two-level indexing for high-dimensional range queries in peer-to-peer networks. In *Proc. of MMSP*, 2009.

[54] M. Zhang and J. Q. Yang. A multi-dimensional query scheme in structured overlays. *Key Engineering Materials*, 2010.

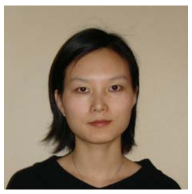
[55] X. Zhang, Y. Qu, and L. Xiao. Improving distributed workload performance by sharing both cpu and memory resources. In *Proc. of ICDCS*, pages 233–241, 2000.

[56] Y. Zhang, L. Liu, D. Li, and X. Lu. DHT-based range query processing for web service discovery. In *Proc. of ICWS*, 2009.

[57] Z. Zhang, S.-M. Shi, and J. Zhu. SOMO: Self-organized metadata overlay for resource management in P2P DHT. In *Proc. of IPTPS*, 2003.

[58] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *J-SAC*, 12(1):41–53, 2004.

[59] C. Zheng, G. Shen, S. Li, and S. Shenker. Distributed segment tree: Support of range query and cover query over DHT. In *Proc. of IPTPS*, 2006.



**Haiying Shen** Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the Holcombe Department of Electrical and Computer Engineering at Clemson University. Her research interests include distributed and parallel computer systems and computer networks, with an emphasis on peer-to-peer and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010 and a member of the IEEE and ACM.

0.2in

**Cheng-Zhong Xu** Cheng-Zhong Xu received B.S. and M.S. degrees from Nanjing University in 1986 and 1989, respectively, and a Ph.D. degree in Computer Science from the University of Hong Kong in 1993. He is currently a Professor in the Department of Electrical and Computer Engineering of Wayne State University and the Director of Sun's Center of Excellence in Open Source Computing and Applications. His research interests are mainly in distributed and parallel systems, particularly in scalable and secure Internet services, autonomic cloud management, energy-aware task scheduling in wireless embedded systems, and high performance cluster and grid computing. He has published more than 160 articles in peer-reviewed journals and conferences in these areas. He is the author of Scalable and Secure Internet Services and Architecture (Chapman & Hall/CRC Press, 2005) and a co-author of Load Balancing in Parallel Computers: Theory and Practice (Kluwer Academic/Springer, 1997). He serves on five journal editorial boards including IEEE TPDS and JPDC. He was a program chair or general chair of a number of conferences, including Infoscale'08, EUC'08, and GCC'07. He is a recipient of the Faculty Research Award of Wayne State University in 2000, the President's Award for Excellence in Teaching in 2002, and the Career Development Chair Award in 2003. He is a senior member of the IEEE.