J. Parallel Distrib. Comput. 69 (2009) 197–209

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A P2P-based intelligent resource discovery mechanism in Internet-based distributed systems

Haiying Shen

Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, United States

ARTICLE INFO

Article history:
Received 13 February 2008
Received in revised form
14 May 2008
Accepted 5 June 2008
Available online 27 June 2008

Keywords:
Resource discovery
Internet-based distributed systems
Peer-to-peer
Distributed hash table
Grids

ABSTRACT

Internet-based distributed systems enable globally-scattered resources to be collectively pooled and used in a cooperative manner to achieve unprecedented petascale supercomputing capabilities. Numerous resource discovery approaches have been proposed to help achieve this goal. To report or discover a multi-attribute resource, most approaches use multiple messages, with one message for each attribute, leading to high overhead of memory consumption, node communication, and subsequent merging operation. Another approach can report and discover a multi-attribute resource using one query by reducing multi-attribute to a single index, but it is not practically effective in an environment with a large number of different resource attributes. Furthermore, few approaches are able to locate resources geographically close to the requesters, which is critical to system performance. This paper presents a P2P-based intelligent resource discovery (PIRD) mechanism that weaves all attributes into a set of indices using locality sensitive hashing, and then maps the indices to a structured P2P overlay. PIRD can discover resources geographically close to requesters by relying on a hierarchical P2P structure. It significantly reduces overhead and improves search efficiency and effectiveness in resource discovery. It further incorporates the Lempel–Ziv–Welch algorithm to compress attribute information for higher efficiency. Theoretical analysis and simulation results demonstrate the efficiency of PIRD in comparison with other approaches. It dramatically reduces overhead and yields significant improvements on the efficiency of resource discovery.

1. Introduction

Advancements in technology over the past decade are leading to a promising future for computing, where globally-scattered resources such as computing resources and data resources are collectively pooled and used in a cooperative manner to achieve unprecedented petascale supercomputing capabilities. Internet-based distributed systems, such as grid and peer-to-peer (P2P) infrastructures, interconnect computers, clusters, storage systems, instruments and so on to make possible the sharing of resources such as CPU time, storage, memory, network bandwidth, and data. Internet-based distributed applications, such as data sharing, computational grids, navigation systems, multimedia and telecommunications, have been widely used in scientific, engineering and commercial areas. For example, the average simultaneous global P2P application users are already over 9 million [3], and BitTorrent P2P file sharing system constitutes roughly 35% of all traffic on the Internet [6].

A fundamental problem in these large, decentralized, distributed resource sharing environments is efficient discovery of resources described by a set of attributes such as CPU speed, memory, and OS version. Another challenge comes from the complex environment characterized by large scale, dynamism, and geographically scattered resources. In such an environment, millions of heterogeneous resources are scattered across geographically distributed nodes, resource utilization and availability are continuously changing, and nodes can enter or leave the system unpredictably.

Centralized resource discovery approaches [14,26,17,10,4,27] are insufficient to deal with these characteristics due to the problem of a single point of failure and bottleneck. More and more approaches resort to structured peer-to-peer (P2P) overlay for resource discovery [1,5,40,31,7,8,28] due to its scalability, efficiency, reliability, self-organization and dynamism-resilience features.

A basic function of a resource discovery technology is to marshal resource information for searching. Current P2P-based methods can be classified into three categories based on the methods of information marshalling. One group of methods [1, 5,40,31] use multiple P2P overlays with one overlay responsible for each attribute name such as CPU and memory, and use the attribute values or attribute descriptions as keywords to store the information in multiple overlays. For example, for a resource

E-mail address: hshen@uark.edu.

0743-7315/$ – see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2008.06.010

"CPU = 2 GHz and Memory = 512 MB", these methods use two messages to store the information of this resource in two overlays responsible for CPU and memory by taking 2 GHz and 512 MB as keywords, respectively. However, multiple P2P overlays require high maintenance overhead, especially in dynamism. Another group [7,8,28] takes both attribute name and value as keywords (i.e. CPU, 2 GHz, Memory and 512 MB), and use four messages to store the resource information in a single P2P overlay. However, it could lead to imbalance of information and load distribution since all resource information of one resource attribute name is accumulated in a single node. For a resource query, the two groups of approaches use multiple queries and present one query for each keyword and then concatenate the results in a database-like "join" operation. However, their efficiency is significantly degraded due to separating a resource description into a number of keywords. First, they lead to high overhead for information storage, reporting, searching and subsequently merging operation. A resource with m keywords in the description needs m messages to store its information in m places. Later, a requester querying for such a resource also needs m messages to search the resources, with one message for each keyword, and then merges a tremendously high volume of information to derive the outcome of the query. To avoid attribute splitting, another class of approaches [34] searches multi-attribute resource using one query by a dimension reducing scheme that reduces multi-attribute to one index. However, it assumes a small number of attributes, and is not effective in a real environment with a tremendously large number of resource attributes. In addition to inefficiency and ineffectiveness, few approaches exploit proximity-aware searching to discover geographically close resources to requesters which is critical to high system performance.

This paper proposes a P2P-based intelligent resource discovery mechanism (PIRD) that weaves all attributes into a set of indices and maps the indices to one P2P overlay using locality sensitive hashing (LSH) [19,18]. PIRD is object-oriented in that it regards the description of a resource such as a computer or a file as a whole entity. Rather than splitting multiple attributes of a resource description, it conducts resource information reporting and searching by taking the multiple attributes as an entire object. PIRD significantly reduces the overhead of resource information reporting, searching and storage, and improves resource discovery efficiency in terms of the number of messages and nodes involved. In addition, by taking advantage of the hierarchical structure of a structured P2P overlay, PIRD can find resources geographically close to requesters. PIRD further uses the Lempel–Ziv–Welch (LZW) algorithm [41] to compress attribute information to reduce overhead and improve efficiency.

The rest of this paper is structured as follows. Section 2 presents a concise review of representative resource discovery in Internet-based distributed systems. Section 3 presents the PIRD mechanism including P2P overlay introduction, LSH and LZW algorithms. Section 4 shows the performance of PIRD using a variety of metrics, and analyzes the factors effecting resource discovery performance. Section 5 concludes this paper.

2. Related work

There have been numerous resource discovery approaches in Internet-based distributed systems. Systems such as Condor-G [15] uses the Globus toolkit [14] to integrate with a grid computing environment for resource management. A number of projects, including Condor [26], XtremWeb [17], Entropia [10], AppLes [4], and Javelin++ [27], have investigated resource searching for computations on grid systems. However, relying on centralized or hierarchical-based policies, these systems have limitations in a dynamic multi-domain environment with variation of resource availability and the presence of large-scale heterogeneity. To cope with these problems, more and more distributed systems resort to structured P2P middleware overlays for resource discovery due to their scalability, reliability and dynamism-resilience.

Structured P2P overlays is an important class of the P2P overlay networks that map keys to the nodes of a network based on a consistent hashing function [21]. Structured P2P overlay is also called Distributed Hash Table (DHT) overlay. Representatives of the structured P2P overlays include CAN [29], Chord [38], Pastry [33], Tapestry [44], Kademlia [25], Symphony [24], Bamboo [32] and Cycloid [36].

To achieve multi-attribute range-query resource discovery, some systems adopt multiple overlays with one overlay responsible for each attribute, and process multi-attribute queries in parallel in corresponding P2P overlays [1,5,40,31]. Andrzejak and Xu [1] extend the CAN-based DHT-system into an indexing infrastructure which allows querying of ranges and supports efficient handling of dynamic data. Furthermore, they use expressway routing [42] in CAN to further cut down costs of searching and updating. The work provides foundation for a self-organizing and scalable implementation of a grid information infrastructure as grid index information service [11], which provides a coherent image of distributed grid resources and allows searching for specific resources. Mercury [5] is a scalable protocol for supporting multi-attribute range-based searches. It differs from other range-based query systems in that it supports multiple attributes as well as performs explicit load balancing. To guarantee efficient routing and load balancing, Mercury uses novel light-weight sampling mechanisms for uniformly sampling random nodes in a highly dynamic overlay network. In Mercury, each query is a conjunction of ranges in one or more attributes. Mercury handles multi-attribute queries by creating a routing hub for each attribute in the application schema. Each routing hub is a logical collection of nodes in the system. Queries are passed to exactly one of the hubs corresponding to the attributes that are queried, while a new data item is sent to all hubs for which it has an associated attribute. This ensures that queries retrieve all relevant data items present in the system. Furthermore, for supporting range queries, Mercury organizes each routing hub into a circular overlay of nodes and places data contiguously on this ring, i.e., each node is responsible for a range of values for the particular attribute. While the notion of a circular overlay is similar in spirit to some existing DHT designs, due to placing data contiguously to support range queries, data partitioning among nodes can become non-uniform. Thus, explicit load-balancing mechanism was further developed to balance the load between nodes. Talia and Trunfio [40] proposed a DHT-based framework, the goal of which is two-fold: to address discovery of multiple resources, and to support discovery of dynamic resources and arbitrary queries in grids. The framework introduced three type of queries: (1) Exact match query, where attribute values of numeric, boolean, or string types are searched; (2) Range query, where a range of numeric or string values are searched; (3) Arbitrary query, where partial phrase match or semantic search is carried out. The framework uses flooding method for resource discovery of dynamic grid resources and for arbitrary query. Because DHT overlays only support exact match lookups, Ratnasamy [31] proposed to construct a Prefix Hash Tree (PHT) to support range queries over DHT overlays. PHT overlays use the hash-table interface of DHT overlays to construct a search tree that is efficient and robust.

Though these methods have different features such as range query, high efficiency and robustness, depending on multiple P2P overlays for multi-attribute resource discovery leads to high maintenance overhead for P2P overlay structures.

Another group of approaches [7,8,28] organize all resource information into one structured P2P overlay and arrange a node be responsible for all information of resources with

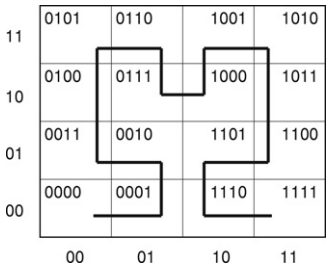


Fig. 1. An example of SFC.

the same attribute. Chord offers efficient and scalable single key-based registration and lookup service for decentralized resources. However, it can not support range queries and multi-attribute-based lookups. MAAN [7] addresses this problem by extending Chord with locality preserving hashing and a recursive multidimensional query resolution mechanism. For attributes with numerical values, MAAN uses locality preserving hashing functions to assign each attribute value an identifier in the m-bit space, and then maps the value information to Chord. Cai and Hwang [8] proposed a scalable grid monitoring architecture that builds distributed aggregation trees (DAT) on Chord. By leveraging Chord topology and routing mechanisms, the DAT trees are implicitly constructed from native Chord routing paths without membership maintenance. To balance the DAT trees, they proposed a balanced routing algorithm on Chord that dynamically selects the parent of a node from its finger nodes by its distance to the root. Oppenheimer et al. [28] developed SWORD which is a Scalable Wide-Area Overlay-based Resource Discovery service. SWORD has a scalable and distributed query processor for satisfying the multi-attribute range queries that describe application resource requirements. It also has techniques for passively and actively balancing load in the range query infrastructure to account for skewed values in measurements.

This group of approaches could result in load imbalance among nodes, and lead to high cost for searching resource information among a huge volume of data in a single node. In most of these works, attributes of a resource are separated and the resource information is reported and stored in a P2P node indexed by each attribute. When a requester searches a resource, it searches each attribute of the resource and then merges the information. For an m-attribute resource, these approaches need m reporting messages, memory size for storing m pieces of information, and m queries for a query, leading to a high number of messages and routing nodes involved and high cost for storage and information merging.

Hilbert space-filling curve (SFC) [2,42] is the main technique for dimension reduction while still preserving the relative distances among points in a multidimensional space. It maps points in an m-dimension Cartesian space into a domain of real numbers; That is, $R^m \longmapsto R^1$, such that the closeness relationship among the points is preserved. This mapping can be regarded as filling a curve within the m-dimensional space until it completely fills the space. Fig. 1 shows an example of SFC. The m-dimensional space is partitioned into 2^{mx} grids of equal size (where m refers to the number of landmarks and x controls the number of grids used to partition the landmark space), and each node is numbered according to the grid into which it falls. This number is called the *Hilbert number* of a node. The Hilbert number indicates the closeness of two points in the space. The smaller the x, the larger the likelihood that two points will have the same Hilbert number.

Based on SFC's feature of locality preserving, Schmidt and Parashar [34] proposed a dimension reducing indexing scheme relying on the SFC that maps the multidimensional information space to P2P nodes, and proved the effectiveness of the SFC adoption for resource discovery in 3-dimensional space. They also indicated that SFC's effectiveness will be degraded with increasing dimension. This is the inherent feature of SFC. Intuitively, higher dimension will make it harder to preserve the locality relationship between points. Though the proposed scheme guarantees that all existing data elements that match a query are found with bounded costs in terms of the number of messages and nodes involved, it is not effective when there are a large number of attributes because of the degrading performance of the dimension reduction algorithm in a high-dimensional space.

In practice, there are a huge number of different resources including data resources such as files, videos and audios, computing resources such as CPU time, storage. Therefore, the number of keywords used to describe millions of various resources is enormously high. Hence, SFC cannot be practically applied to resource discovery where there are significantly large number of keywords.

More importantly, most current P2P-based methods are unable to discover resources geographically close to the requester, which is very important for resource sharing performance. Unlike the current three groups of approaches, the PIRD mechanism is object-oriented by regarding all attributes of a resource as an entire object. Without separating attributes in a resource, it weaves all attributes into a set of indices and maps the indices to one P2P overlay for efficient resource discovery. PIRD achieves balanced load distribution with low overlay structure maintenance overhead. Furthermore, by taking advantage of P2P hierarchical structure, PIRD clusters the resource information based on the proximity of resource so that it can find geographically close resources to requesters for high system performance.

3. PIRD: P2P-based intelligent resource discovery

PIRD is built on top of a single hierarchical Cycloid structured P2P overlay network [36] to achieve multi-attribute and proximity-aware resource discovery. PIRD can also use a flat structured P2P overlay [36,38,44,33,29] such as Chord as a underlying overlay; its details will be presented in Section 3.2. Before we begin more detailed discussion of PIRD, we briefly describe structured P2P overlay networks and Cycloid. Structured P2P overlay networks is a class of decentralized systems in the application-level that partition ownership of a set of objects among participating nodes. The overlay networks can efficiently route messages to the unique owner of any given object. Each object or node is assigned an ID (i.e., key) that is the hashed value of the object or node IP address using consistent hash function [21]. The overlay network provides two main functions: `Insert(ID,object)` and `Lookup(ID)` to store an object to a node responsible for the ID, and to retrieve the object.

Cycloid is a lookup efficient overlay. It can have maximum $n = d \cdot 2^d$ nodes with d as its dimension. An ID of a Cycloid node or object is represented by a pair of indices

$$ID = (ID_{cyc}, ID_{cub}) = (k, a_{d-1}a_{d-2}\ldots a_0),$$

where k is a cyclic index and $a_{d-1}a_{d-2}\ldots a_0$ is a cubical index, represented by ID_{cyc} and ID_{cub}, respectively. The ranges of the cyclic index and cubical index are $[0, d-1]$ and $[0, 2^d-1]$, respectively. Given a Cycloid P2P overlay of $d\cdot 2^d$ nodes, the domain of cyclic indices consists of d number, i.e., $\{0, 1, \ldots, d-1\}$, and the domain of cubical indices consists of 2^d number, i.e., $\{0, 1, \ldots, 2^d-2, 2^d-1\}$. The right side of Fig. 5 shows the partial routing links of a 11-dimensional Cycloid, where x indicates all possible cyclic indices. The nodes with the same cubical index are ordered by their ID_{cyc} mod d on a small cycle, which is called *cluster*. All clusters are ordered by their ID_{cub} mod 2^d on a large cycle. Thus, the nodes in Cycloid are grouped into different clusters, which are identified by the cubical indices. Within a cluster, the nodes are differentiated by

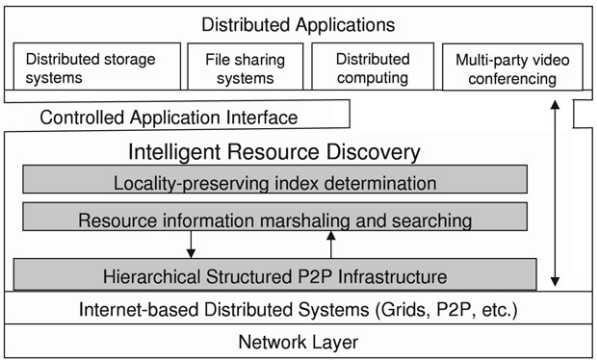


Fig. 2. High-level architecture of PIRD.

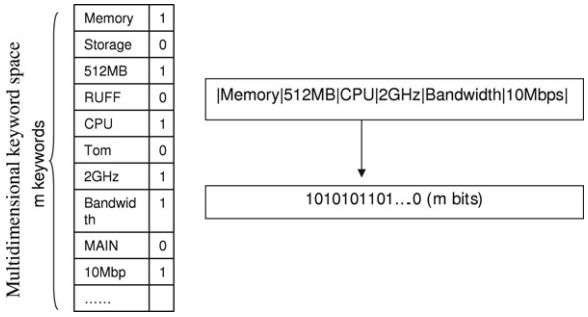


Fig. 3. Resource vector generation.

the cyclic indices. Cycloid assigns a key to a node with the closest ID to the key's ID. For more information about Cycloid, please refer to [36].

Fig. 2 shows a high-level view of the architecture of PIRD. PIRD includes two characteristic components to efficiently and effectively search resources for high performance Internet-based distributed systems. The two components are locality-preserving index determination and resource information marshaling and searching.

Locality-preserving index determination. The fundamental functionality of this component is to represent each resource and query by a set of indices that preserve locality. Particularly, a resource or query is described by a vector where each dimension is associated with a distinct keyword. Usually, a resource attribute has a keyword to represent itself. Resources without keywords such as a file can use information retrieval methods [12] to get its vector. Similar resources are considered to have similar vectors. The resource vector is then used to produce a small set of indices through LSH. Resources and queries with similar vectors will have similar indices.

Resource information marshaling and searching. This component provides resource reporting and retrieval capabilities. The functionality of reporting is to store the information of each resource automatically in a structured P2P overlay according to resource attribute and proximity. The functionality of resource searching is to locate desired resources for a given query in a distributed and efficient manner. The P2P overlay lookup function facilitates requesters to discover resources efficiently, and the hierarchical overlay structure further enables requesters to locate geographically close resources for high performance.

3.1. Locality-preserving index determination

A key component of a resource discovery system is defining an index space and deterministically mapping resources to this index space. To support complex resource searches in a resource discovery system, PIRD associates each resource with a set of keywords and defines a mapping that preserves the similarity among the keyword sets of different resources.

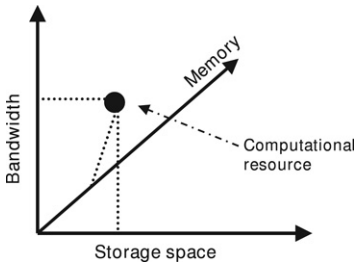


Fig. 4. A 3-D keyword space.

The keywords are common words to describe resource attributes such as bandwidth and memory, and values of globally defined resource attributes such as 10 Mbps and 512 MB. As a result, each resource is represented by a keyword vector. All keywords in the system form a multidimensional keyword space where resources are points in the space and the keywords are the coordinates. Fig. 4 shows an example of a 3-dimensional keyword space [34]. The keywords can be viewed as base-2^m numbers, where m is the total number of keywords in the resource discovery system. Two resources are considered "local" if they are close together in this keyword space. For example, "Memory 512 MB CPU 2 GHz" and "Memory 1 GB CPU 2 GHz" are local since they have many common keywords. Fig. 3 demonstrates how a resource's vector is determined. If a resource has a keyword in a dimension of the m-dimensional keyword space, it has "1" in that dimension. Otherwise it has "0" in that dimension. Finally, a resource gets a resource vector with length of m bits.

The problem of resource discovery can be regarded as finding the nearest neighbor of a query point in a high dimensional keyword space mainly focusing on the Euclidean space with Euclidean distance as metric: given n points in an m-dimensional space, find the nearest neighbor of a query point. The next question is how to transform resource vectors to indices in an index space. To efficiently support queries using partial keywords, the index space should preserve the locality of the points so that located points can be refined by setting different neighbor distance ranges. Locality-sensitive hashing (LSH) [19,18] is the main technology for dimension reduction while still preserving the relative distances among points in a multidimensional space.

Locality Sensitive Hashing. LSH is locality sensitive in that is maps points in an m-dimensional space into one-dimensional space while still preserving the closeness relationship among the points. LSH is an algorithm for solving approximate and exact near neighbor search in high dimensional spaces. The intuition of LSH is: if two points are close (less than distance r_1), they hash to same value with probability of at least p_1; if they are far away between each other (more than distance $r_2 > r_1$), they hash to same value with probability of no more than $p_2 < p_1$. Specifically, for a domain S of the points set and distance measure D, the LSH family is defined as follows, where U is an index space domain, $B(q, r)$ represents the scope with range r around point q, and **Pr** denotes probability.

Definition 1. A family $\mathcal{H} = \{h : S \rightarrow U\}$ is LSH functions for distance function D if for any two points $\mathbf{v}, \mathbf{q} \in S$
- if $\mathbf{v} \in B(q, r_1)$ then $\mathbf{Pr}_{\mathcal{H}}[h(q) = h(v)] \geq p_1$,
- if $\mathbf{v} \notin B(q, r_2)$ then $\mathbf{Pr}_{\mathcal{H}}[h(q) = h(v)] \leq p_2$,
- $r1 < r2, p1 > p2$.

LSH-based resource index determination. In the following, we discuss how to transform resource vectors to indices so that similar resources have similar indices with high probability.

Different LSH families can be used for different distance functions. PIRD relies on the LSH technique in Euclidean spaces [16].

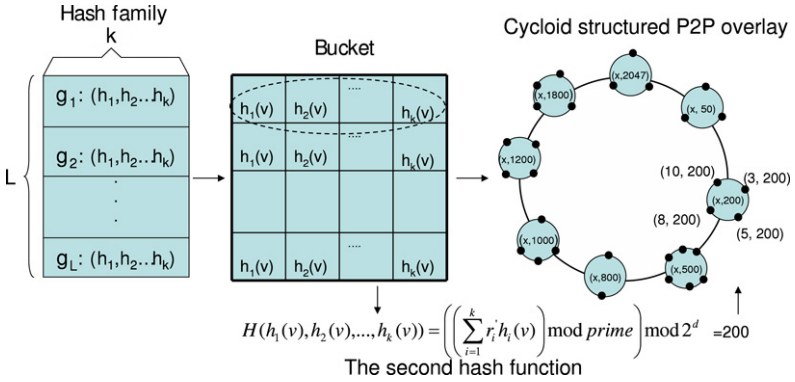


Fig. 5. Process of ID mapping in PIRD.

Based on this technique, a family $\mathcal{H}$ of hash functions is derived. That is, $h_{a,b}(v) = [\frac{a \cdot v + b}{w}]$, where a is an m-dimensional random vector whose each entry is drawn from a Gaussian distribution $Ga(x)$ (x is generated randomly in [0,1]), w is a specified integer value, and b is a random real number chosen uniformly from [0, w]. PIRD defines a function family $G = \{g : S \rightarrow U^k\}$ such that $g(v) = (h_1(v), \ldots, h_k(v))$, where h_i belongs to $\mathcal{H}$, and k is a specified integer value. k represents a tradeoff between memory consumption for storing resource information and time consumption for pruning false positives. False positives are discovered points that actually are not close to a query point. A false positive incurs when LSH generates the same hash value for two points that are not close. Larger k leads to more memory consumption (also longer time spent in computing hash values), but fine-grained points and hence less false positives. On the other hand, smaller k leads to less memory consumption (also shorter time spent in computing hash values), but coarse-grained points and hence more false positives.

Fig. 5 illustrates the mapping of a resource to a Cycloid P2P node in PIRD. A *bucket* is used to store a group of hashed values generated by a $g(v)$. In other words, the hashed values of a resource vector v, $h_i(v)(1 \leqslant i \leqslant k)$, is stored in a bucket. Thus, L number of $g(v)$ leads to L buckets, where L is a specified constant. Then PIRD uses another hash function H:

$$ID_{lsh} = H(h_1(v), h_2(v), \ldots, h_k(v)) = \left(\left(\sum_{i=1}^{k} r_i' h_i(v)\right) \bmod \text{prime}\right) \bmod 2^d$$

to compute the index of each bucket, denoted by ID_{lsh}. In the hash function, 2^d represents the range of Cycloid cubical indices ID_{cub}, *prime* denotes a prime number, and r' is a random 32-bits unsigned integer number randomly generated in the range of $[1, 2^{29}]$ [23]. r' is fixed for all buckets. The indices of a resource vector v are its cubical indices of its final Cycloid IDs for mapping to Cycloid P2P overlay. In conclusion, given a resource vector v, PIRD hashes v to L buckets of k hash values, and calculates the final L hash values through the hash function H on each of L buckets. Finally, similar resource vectors have similar ID_{lsh}.

3.2. Resource information marshalling and searching

After the indices of a resource vector are determined, the next question is how to map the resource vector to a structured P2P overlay for efficient multi-attribute and proximity-aware resource searching. Locating geographically close resources is important to the performance of Internet-based distributed applications, especially to time-critical applications, since some tasks may suffer from long distance delay. PIRD marshals the information of physically close resources in one node to achieve proximity-aware resource discovery. Proximity information generation is necessary to achieve this goal. In the following, we first introduce the proximity information generation method to represent node closeness on the Internet by indices. We then introduce the algorithms for resource information marshalling and searching.

Proximity information generation. Landmark clustering has been widely adopted to generate proximity information [30,42,37]. It is based on the intuition that nodes close to each other are likely to have similar distances to a few selected landmark nodes, although details may vary from system to system. In P2P overlays, the landmark nodes can be selected by the overlay itself or the Internet. More sophisticated strategies [20,9,13] can be used for landmark nodes selection. We use a simple method with a distance constraint between landmarks. We assume $m = 14$ landmark nodes that are randomly scattered in the Internet. Each node measures its physical distances to the m landmarks, and use the vector of distances $\langle d_1, d_2, \ldots, d_m \rangle$ as its coordinate in Cartesian space. Two physically close nodes will have similar landmark vectors. We use Hilbert space-filling curve (SFC) [2,42] to map m-dimensional landmark vectors to Hilbert numbers. SFCs map points in an m-dimensional Cartesian space into a domain of real numbers. That is, $R^m \longmapsto R^1$, such that the closeness relationship among the node points is preserved. We use $\mathbb{H}$ to denote the Hilbert number of a node. The Hilbert number indicates physical closeness of nodes on the Internet. For instance, if the relationship of the Hilbert numbers of nodes i, j and k is $\mathbb{H}_i < \mathbb{H}_j < \mathbb{H}_k$, then node j is geographically closer to node k than node i.

Resource information marshalling algorithm. Recall that Cycloid consists of a number of clusters, which constitute a large cycle. PIRD lets each cluster be responsible for the information of similar resources, and divides the resource information among nodes within the cluster based on geographical closeness of the resources. In a Cycloid ID, the cubical indices differentiate clusters, and the cyclic indices differentiate nodes in a cluster. PIRD uses cubical indices to represent different resources, and uses cyclic indices to represent the locations of resource host nodes. Specifically, it defines a resource's P2P identifier as $ID = (\mathbb{H}, ID_{lsh})$, where $\mathbb{H}$ is the Hilbert number of the resource host node. Recall that a P2P overlay network provides two main functions: `Insert(ID,object)` and `Lookup(ID)` to store an object in a node responsible for the ID, and to retrieve the object. PIRD lets nodes report their resource information to the system by `Insert` $(ID, \langle v, ip\_addr \rangle)$, where $\langle v, ip\_addr \rangle$ represents the resource information. Based on Cyloid topology and key assignment algorithm, the information of similar resources will be stored in the same cluster. Within each cluster, the information of resources in close proximity will gather together in a node. We call the repository node *directory node*. For example, in Fig. 5, one of the resource's ID_{lsh} is 200, then the information of this resource is mapped to cluster with ID_{cub} equals to 200. Assume the Hilbert number of the resource's host node is 5, then the host node

of the resource uses the function Insert ((5, 200), ⟨v, *ip_addr*⟩), and the information ⟨v, *ip_addr*⟩ will be stored in node (5, 200).

Theorem 3.1. *If nodes report their resource information by targeting* ($\mathbb{H}$, ID_{lsh})*, the information of similar resources will be stored in the same cluster, and within each cluster, the information of resources in close proximity will be stored in one node.*

Proof. In a Cycloid ID (ID_{cyc}, ID_{cub}), cubical indices differentiate clusters, while the cyclic indices differentiate nodes in one cluster. In ID ($\mathbb{H}$, ID_{lsh}), the ID_{lsh} is determined by resource type, so similar resources will be in a same cluster. Further, the $\mathbb{H}$ indicates the proximity closeness of resource hosts, hence the information of resources in close proximity will gather together in one node. ■

Specifically, given a resource with a keyword vector v and its host node's IP address, the PIRD maps v into L indices. Each of the L indices (possibly duplicated) corresponds to a cubical index. The physical position of the resource host node is mapped into a Hilbert number, which corresponds to the cyclic index of the directory node for the given resource. It implies that the L directory nodes for a resource have the same cyclic index. For instance, a resource of "Memory 512 MB CPU 2 GHz Bandwidth 10 Mbps" gets L indices,

$$ID_{lsh_1}, ID_{lsh_2}, \ldots, ID_{lsh_L}$$

using the LSH-based ID determination method. It then generates L Cycloid IDs

$$ID_1, ID_2, \ldots, ID_L, \quad \text{where } ID_i = (\mathbb{H}, ID_{lsh_i}).$$

After that, the node uses

$$\texttt{Insert}(ID_1, \langle v, ip\_addr\rangle), \ldots, \texttt{Insert}(ID_L, \langle v, ip\_addr\rangle)$$

to insert its resource information into L nodes in Cylcoid P2P overlay. Algorithm 1 shows the pseudo-code of resource information marshalling in PIRD.

Algorithm 1: Pseudo-code for resource information marshalling.

```
//convert a resource vector v to a set of ID_lshs
for each g[j] do { //g[j] is one of L groups of hash functions
   ID_lsh[j] = 0
   for each h[i] in g[j] do {//g[j] has k hash functions
      ID_lsh[j]+ = (r'_i h[i](v) mod prime) mod 2^d }
}
ℍ=the Hilbert number of the host node of the resource
for each ID_lsh[j] do {
   //insert the resource information ⟨v, ip_addr⟩ into the P2P
   overlay
   ID[j] = (ℍ, ID_lsh[j])
   Insert(ID[j],< v,ip_addr>)
}
```

Note that for two resource vectors v_1 and v_2,

$$\mathbf{Pr}_{h\in\mathcal{H}}[h(v_1) = h(v_2)] = \text{sim}(v_1, v_2),$$

where "sim(v_1, v_2)" represents the similarity between v_1 and v_2. For L different groups $g_1, g_2, \ldots, g_L$ of hash functions, the probability that v_1 and v_2 cannot produce the same indices for all k hash functions $\in g_i$ is $1 - p^k$, where p denotes sim(v_1, v_2). The probability that v_1 and v_2 can produce the same index for all k hash functions of at least one of L groups is $1-(1-p^k)^L$. In other words, the resource information marshalling can have resource vectors with similarity p hashed to the same nodes with a probability no less than $1-(1-p^k)^L$. For example, if two resource vectors v_1 and v_2 have a similarity $p = 0.7$, $k = 5$ and $L = 5$, the probability of hashing the two vectors to at least one same node is 60% [45].

Resource information searching algorithm. We now discuss the issue of how to locate resources for a resource query, given the fact that all resources in the system are indexed to Cycloid overlay according to their keyword vectors. The goal of PIRD is to answer a query by consulting only a small number of nodes that are most responsible for the query.

As described above, a resource in the system is associated with a sequence of one or more keywords. The queries can consist of a combination of keywords or partial keywords. For example, (computer, network) and (computer, *) are all valid queries. A query with partial keywords may include a huge number of possible vectors. However, no matter how many possible vectors there are, the PIRD will always generate only L hash values for the query, and will locate all resources with keywords similar to the query. The expected result of a query is the information of the complete set of resources matching the user's query.

Processing a query consists of two steps: translating the keyword query to relevant Cycloid P2P IDs, and querying the appropriate nodes in the overlay network for resource information. Recall that a piece of resource information is stored in L directory nodes responsible for ($\mathbb{H}$, ID_{lsh_i})($1 \le i \le L$) in the structured P2P overlay. Therefore, when discovering a resource, L resource IDs should be generated for the queried resource using the locality-preserving index determination method and proximity information generation method. The resource IDs for a resource query is ($\mathbb{H}$, ID_{lsh_i})($1 \le i \le L$), where $\mathbb{H}$ is the Hilbert number of the requester node. The destination of the L IDs should be queried for the resource information. Hence, when a node queries for different resources, it sends out different requests using Lookup ($\mathbb{H}$, ID_{lsh}) for each resource ID. A query message targeting each ID is routed to the destination node based on structured P2P lookup algorithm: given an ID for an object, the node responsible for the object is located. The destination node is the directory node responsible for the information of the resource in geographically close proximity of the requester. No matter whether a query consists of all keywords or partial keywords, it will be mapped to at most L points in the ID space, which means that at most L nodes will be visited for resource information. Finally, the nodes containing the information of the queried resource receive the requests.

Algorithm 2: Pseudo-code for resource information searching.

```
//convert the query resource vector v to a set of ID_lshs
for each g[j] do {//g[j] is one of L groups of hash functions
   ID_lsh[j] = 0
   for each h[i] in g[j] do {//g[j] has k hash functions
      ID_lsh[j]+ = (r'_i h[i](v) mod prime) mod 2^d }
}
ℍ=the Hilbert number of the requester
for each ID_lsh[j] do {
   //send a request to a node which is the destination of the ID[j] in
   the P2P overlay
   ID[j] = (ℍ, ID_lsh[j])
   send Lookup(ID[j])
}
receive responses from all the destination nodes
merge the response information that satisfies the query-based
on ip_addr
prune the resource vectors whose vectors satisfy d(v, q) > r
```

For example, q is a resource vector for a resource query. PIRD produces L P2P IDs from q using the same set of LSH hash functions. Therefore, if a resource satisfies the query, its information will be retrieved with very high probability. Note that the vectors of resources and the query could be hashed to the same ID_{lsh} with high probability (i.e., $1-(1-p^k)^L$). Thus, by having these ID_{lsh}s as the P2P cubical IDs in the structured P2P overlay's function Lookup(ID) (ID = ($\mathbb{H}$, ID_{lsh})), PIRD is able to retrieve desired resources from the directory nodes that are responsible for these IDs. For example, in Fig. 5, if one generated ID of a query is (5, 200),

node (5, 200) will receive a Lookup request. L is very small (e.g., 5) in our system, which implies that a query can be answered by consulting only a small number of nodes no more than L.

Upon receiving a request, each destination node checks locally the list of tuples $\langle v, ip\_addr\rangle$ and returns the ip_addr of nodes that have resources similar to the query's v. Then, the requester merges the replies from all destination nodes based on ip_addr. Let $v_1, \ldots, v_t$ be the resource vectors after merging, PIRD then computes the Euclidean space distance between q and $v_1, \ldots, v_t$ using

$$d(v, q) = \|v - q\| = \sqrt{\sum_{i=1}^{m}(v_i - q_i)^2},$$

where v_i and q_i represents a value in one dimension, and r is a pre-defined threshold of Euclidean distance between a neighbor point and a query point. For each v, if v belongs to $B(q, r)$, that is, $d(v, q) \leq r$, then v is a neighbor point in range r, i.e, a desired resource. Otherwise, v is not a resource that satisfies query q. This refinement step is to prune false positive results. The resource requester then requests resources from resource hosts identified by ip_addr. Algorithm 2 shows the pseudo-code of resource information searching in PIRD.

In addition to a hierarchical structured P2P overlay, PIRD can also work on a flat structured P2P overlay such as Chord. In a flat structured P2P overlay, when a node reports the information of its resources, it also reports its Hilbert number $\mathbb{H}$. Each destination node organizes the information tuples in such a way that the tuples are grouped locally based on their hosts' Hilbert number. When node i queries for a resource, it also sends its Hilbert number $\mathbb{H}_i$ along with the request. After a destination node receives the query, it only needs to check the group whose $\mathbb{H} \approx \mathbb{H}_i$. Consequently, node i receives the information of resources that is located geographically close to it. Communicating with geographically close nodes, and using geographically close resources improve the system performance significantly.

Comparative analysis. The following theorems compare the performance of PIRD with MAAN [7] and Mercury [5] resource discovery methods.

Theorem 3.2. *For a resource with m attributes each of which has k keywords, MAAN and Mercury store the information of the resource in $m \times (1 + k)$ and $m \times k$ P2P nodes, respectively, while PIRD stores the information of the resource in $\leq L$ nodes.*

Proof. For each piece of resource information including attribute name and value (or string description), MAAN splits the keywords and stores the information based on each keyword and attribute name. Thus, it produces $m \times (1+k)$ pieces of resource information in $m \times (1 + k)$ nodes. Mercury depends on multiple P2P overlays with one overlay responsible for each attribute name. It takes attribute descriptions as the key for information reporting. Hence, Mercury generates $m \times k$ messages. Regardless of the length of a resource description, PIRD produces L IDs. Since the IDs might have coincided ones, PIRD leads to $\leq L$ nodes for storing resource information. ■

Corollary of Theorem 3.2. *For a resource with m attributes each of which has k keywords, MAAN and Mercury need no less than $m \times k$ messages to report the resource to or query the resource from directory nodes, while PIRD only needs $\leq L$ messages.*

Proof. To query a resource, all information of the resource in the system should be retrieved and merged. According to Theorem 3.2, the Corollary of Theorem 3.2 is proven. ■

Theorem 3.3. *For any set of n nodes and m resource attributes, with high probability,[1] PIRD can improve the maintenance overhead of resource discovery structure of multi-P2P-based resource discovery methods (e.g. Mercury) by no less than a factor of m.*

Proof. In PIRD, each node is responsible for maintaining approximately $\log(n)$ neighbors. In multiple-P2P-based resource discovery, a node is a member of each of the multiple P2P overlays. Therefore, a node is responsible for maintaining $\log(n)$ neighbors for each P2P overlay. Hence, a node has $m \log(n)$ neighbors. ■

Theorem 3.4. *For any set of n nodes where $n = d \cdot 2^d$, and k pieces of resource information of a resource, with high probability, PIRD can reduce the load imbalance of resource information distribution of single P2P-based resource discovery methods (e.g. MAAN) by a factor of $\frac{2^d}{1+d}$.*

Proof. Consistent hashing is proved to have a bound of $\epsilon = \log(n)$ [38], which means statistically a node can at most be mapped onto by $(1 + \epsilon)\frac{k}{n}$ pieces of resource information with high probability, given k pieces of total resource information of a resource. For any set of n nodes where $n = d \cdot 2^d$, and k pieces of resource information of one resource, a node has k pieces of resource information in single P2P-based resource discovery methods based on Chord. In PIRD, a directory node for the resource in a cluster will have at most $(1 + \log 2^d)\frac{k}{2^d}$ pieces of resource information by regarding each cluster as a node in Chord. We define *load imbalance* as the difference between the heaviest node and lightest node measured by the pieces of resource information. Therefore, PIRD can improve of the load imbalance of resource information distribution of single P2P-based resource discovery methods by a factor of $\frac{k}{(1+\log 2^d)\frac{k}{2^d}} = \frac{2^d}{1+d}$. ■

Theorem 3.5. *For a resource with m attributes each of which has k keywords, MAAN and Mercury need no less than $mk\log(n)$ routing nodes to report the resource to or query the resource from destination nodes, while PIRD only needs $\leq L\log(n)$ routing nodes.*

Proof. In the average case, the lookup path length of Chord [38] and Cycloid [36] is $\log(n)$. Based on the Corollary of Theorem 3.2, it is proven. ■

3.3. Dynamism-resilient resource discovery

In a dynamic environment, nodes join in and leave the system continuously and frequently. An efficient resource discovery mechanism should be able to deal with dynamism. For example, a node departure outdates resource information, or a failed directory node makes the resource information unavailable. PIRD uses Cycloid self-organization mechanism to maintain the middleware architecture and resource information. It deals with node joins and departures in a distributed manner, without requiring information to be propagated through the entire network.

Recall that with the Cycloid's key assignment protocol, the key is stored in its owner that has the closest ID with the key's ID. In the Cycloid self-maintenance mechanism, when a key's owner leaves, it will forward the key to the key's new owner before leaving. When a node joins in the system, it receives the keys from its neighbors that are in its responsible ID region. Therefore, a key is always stored in its owner even in dynamism. Like a key, a resource also has P2P IDs. PIRD transfers resource information along with the keys in dynamism in the same manner as key transfer. That is,

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

resource information is transferred to a node with ID closest to the resource's ID. Hence, resource information is always stored in a directory node in the Cycloid even in a dynamic environment. For example, in Fig. 5, when node (5, 200) leaves the system, it transfers its resource information to nodes (3, 200) and (8, 200) which is the new owner of the information based on the ID closeness. Typically, the resource information in the range of (3, 200) and (5, 200) is transferred to node (3, 200), and the resource information in the range of (5, 200) and (8, 200) is transferred to node (8, 200). If node (5, 200) is the only node in its cluster, it transfers its information to its closest node in its closest cluster. If node (7, 200) joins the system, then the resource information in the range (6, 200) and (7, 200) is transferred from node (5, 200) to the newly-joined node (7, 200). The consistent hashing for key assignment protocol requires relatively little re-assignment of resource information as nodes join and leave the system.

For node failures without warning, PIRD resorts to the periodical resource information reporting by which the lost resource information will be recovered in its new directory node. When a directory node receives a resource request, if it cannot locate requested resource in its own directory, it assumes that the old directory node of the resource information has failed, and waits for a period of time T which is the resource information reporting period. Within the T time period, the lost resource information will be reported to the node. To prevent the information space from being flooded with outdated information left behind by node failures, nodes execute garbage collection periodically. Particularly, after a period of time, if a node has not received resource information from another node, it deletes the information of the node.

In addition to maintaining the resource information, Cycloid self-maintenance mechanism also helps to maintain the Cycloid architecture in dynamism. When a new node joins, it initializes its routing table, and informs other related nodes of its presence. Before a node leaves, it notifies its neighbors. The informed nodes will update their corresponding neighbors. Like other structured P2P overlays, Cycloid relies on stabilization for node failures in which a node periodically updates its neighbors. For more details of the Cycloid self-organization mechanism, please refer to [36].

Consequently, instead of relying on specific nodes for storing resource information, PIRD always stores resource information in directory nodes even in dynamic situation, and the `Lookup` requests will always be forwarded to the directory nodes having the required resource information. As a result, PIRD has high capability to deal with dynamism in Internet-based distributed systems.

3.4. Optimized PIRD

In an Internet-based distributed system with enormous number of keywords, each resource will have a long vector even though it has only a few keywords. Tremendously long vectors with sparse keywords lead to inefficiency of vector processing in PIRD. For instance, a resource query only wants memory and CPU in a 10,000-dimensional keyword space, then its resource vector will have two 1 bits with all other bits equal to 0. PIRD needs to be complemented by a compression algorithm to make it more efficient. This is confirmed by Lemma 3.1. We leave the proofs for the lemmas in this section to Appendix.

Lemma 3.1. *PIRD has higher efficiency on shorter vectors than on longer vectors.*

To optimize PIRD in processing long and sparse vectors, we adopt LZW dynamic compression algorithm [41] to reduce the dimension of vectors and remove insignificant strings, i.e. "0"s.

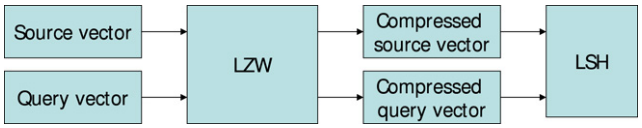


Fig. 6. Integration of LZW into LSH.

LZW is a universal lossless data compression algorithm. It replaces repetitive substrings of characters with *codes*, while still keeping the old string's information in the compressed string. The code consists of digital numbers that can be repetitive. For example, for a string ABCCAABCDDAACCDB, if LZW uses 4 to denote AB and 5 to denote CC, after compression, the string becomes 45A4CDDAA5DB. Fig. 6 shows a model of the integration of LZW algorithm to the LSH process. PIRD first uses LZW to compress long resource or query vectors to shorter vectors before using LSH to calculate the indices. Thus, the efficiency of PIRD is enhanced.

LZW data compression algorithm. LZW encodes data by referencing a string translation table that records the codes with their corresponding substrings. LZW replaces a substring with its corresponding code during compressing.

Algorithm 3: Pseudo-code for the LZW compression algorithm

```
code=0;
//initialize table with each character member
do{
   P=next input character
   if P is not in the translation table then{
      put P into the translation table
      assign code++ to P as its code}
while P is not the end of input string}

//compress the string
P= first input character
while it is not the end of input string do{
   C=next input character
   if P+C is in the string table then
      P=P+C
   else {
      output the code of P
      assign code++ to P+C as its code
      add P+C to the translation table
      P=C}
}
output the code of P
```

Algorithm 3 shows the pseudo-code for the LZW compression algorithm [22]. Fig. 7 demonstrates a simple example of LZW translation table and the process of the LZW algorithm. The input string is "ABABABABBBABABA". The translation table is initialized with each character member of the string as well as "Clear" and "End". "Clear" means reconstructing the dictionary. "End" means the scanning of the string is completed. Each character is assigned a code starting from 0 in an increasing order. The input string has character members "A" and "B". Therefore, the translation table is initialized with "A", "B", "Clear" and "End". "A" is assigned code "0", "B" is assigned "1", and so on. Then, the LZW algorithm examines the string characters serially, and stores every unique two-character (a code is also regarded as a character) string into the translation table and assigns it a new code in an increasing order. Specifically, whenever a previously-encountered string is read from the input, the longest previously-encountered string is determined, and the code for this string concatenated with the extension character (the next character in the input) is stored into the translation table and assigned a new code. The code for the longest previously-encountered string is then output and the

Character	Prefix	Suffix	String	In table	Code	Output
A	--	--	--	--	0	--
B	--	--	--	--	1	--
Clear	--	--	--	--	2	--
End	--	--	--	--	3	--
A	--	A	A	yes	--	--
B	A	B	AB	no	4	A
A	B	A	BA	no	5	B
B	A	B	AB	yes	--	--
A	4(AB)	A	4A(ABA)	no	6	4(AB)
B	A	B	AB	yes	--	--
A	4(AB)	A	4A(ABA)	yes	--	--
B	6(ABA)	B	6B(ABAB)	no	7	6(ABA)
B	B	B	BB	no	8	B
B	B	B	BB	yes	--	--
A	8(BB)	A	8A(BBA)	no	9	8(BB)
B	A	B	AB	yes	--	--
A	4(AB)	A	4A(ABA)	yes	--	--
B	6(ABA)	B	6B(ABAB)	yes	--	--
A	7(ABAB)	A	7A(ABABA)	no	10	7

Fig. 7. Example of the LZW algorithm.

extension character is used as the beginning of the next string. For example, "A" is read from the string at first. Then, "B" is read from the string. The prefix is "A", the suffix is "B", and the concatenated string is "AB". Since "AB" is not in the translation table, "A" is output and "B" becomes the prefix of the next string. Also, "AB" is added to the translation table and assigned code "4". In the next string, the second "A" is read from the string, which is the suffix. Combining the prefix and suffix, "BA" is generated. Since "BA" is not in the translation table, "B" is output and "A" becomes the prefix of the next string. Similarly, "BA" is added into the translation table and assigned code "5". In the subsequent step, "B" is read from the string, which is the suffix. The combined string "AB" is already in the translation table with code "4", then there is no output and "4(AB)" becomes the prefix of the next step. "4(AB)" means "4" is equivalent to "AB". The same to the other parenthesis notations. This process is repeated until the end of the input string is reached. After compression, the string turns to AB46B87.

Analysis of the integration of LZW into LSH. We analyze the effectiveness of the integration of LZW into LSH. *Compression rate* is used to represent the factor that the length of a string is reduced. More repetitive characters or substrings in a string will lead to higher compression rate.

Whether LZW can be incorporated into LSH for effective resource queries is determined by whether LZW is locality preserving; that is, whether the similarity of two strings remains the same after compression. Based on the LZW operation, we proved that LZW is locality preserving as shown in Lemma 3.2.

Lemma 3.2. *The similarity between two strings remains the same before and after the compression using the LZW algorithm.*

Therefore, LZW keeps the locality preserving property of LSH for resource discovery. The next question is whether the adoption of LZW in LSH will lead to improvement on LSH's efficiency. We analyzed the impact of LZW on resource vectors, and concluded that LZW can be exploited to its full capacity on LSH. The details are demonstrated in Lemma 3.3.

Lemma 3.3. *The LZW algorithm has a higher compression rate for a string that contains a larger number of repetitive substrings, and thus leads to a high compression rate for long resource vectors with repetitive "0"s and "1"s.*

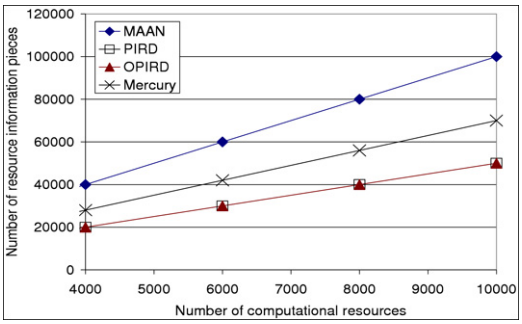
We analyzed the time complexity of PIRD with LZW, and found that LZW can improve the efficiency of LSH dramatically.

Lemma 3.4. *The performance of PIRD is improved after the resource and query vectors are compressed by the LZW algorithm.*

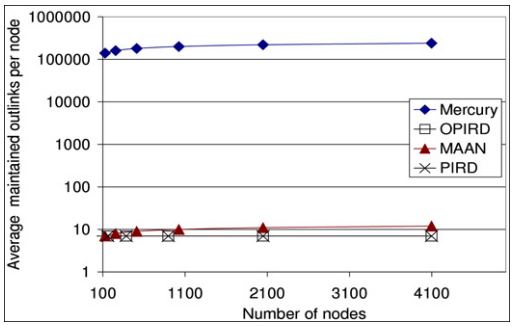
4. Performance evaluation

We designed and implemented a simulator in Java for evaluation of PIRD and optimized PIRD (OPIRD) based on Cycloid hierarchical P2P overlay [36] and E2LSH 0.1 [23]. E2LSH 0.1 is a simulator for the high-dimensional near neighbor search based on LSH in the Euclidean space. We compared the performance of PIRD with MAAN [7] and Mercury [5] on Chord [38], in terms of storage requirement for resource information polling, load balance, structure maintenance cost, the number of nodes involved for information marshalling, resource search latency, and proximity-aware performance for geographical close resource discovery. MAAN relies on one Chord and maps each resource keyword to a P2P node for resource information marshalling. Mercury depends on multiple P2P overlays with each P2P overlay responsible for an attribute name, and maps attribute value to a P2P node for resource information marshalling. We conducted an experiment on SFC [34], and found that SFC is not effective in a large dimension space. All resource vectors are hashed to the same value. This is consistent to the statement in [34] that the SFC works well in finding nearest neighbors in low dimensions but its performance degrades when the number of dimensions increases to a large number.

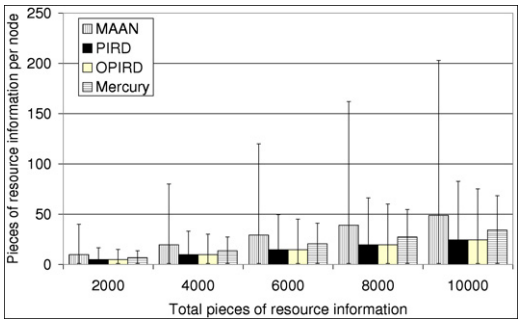
We used two transit-stub topologies generated by GT-ITM developed by Georgia Tech research group [43]. A routing domain in the Internet can be classified as either a stub domain or a transit domain. A stub domain carries only traffic that originates or terminates in the domain. Transit domains do not have this restriction. The purpose of transit domains is to interconnect stub domains efficiently. Stub domains generally correspond to campus networks or other collections of interconnected LANs while transit domains are almost always wide or metropolitan area networks such as WANs and MANs. The two topologies used in the simulation are "ts5k-large" and "ts5k-small" with approximately 5000 nodes each. "ts5k-large" has 5 transit domains, 3 transit nodes per transit domain, 5 stub domains attached to each transit node, and 60 nodes in each stub domain on average. "ts5k-small" has 120 transit domains, 5 transit nodes per transit domain, 4 stub domains attached to each transit node, and 2 nodes in each stub domain on average. "ts5k-large" has a larger backbone and sparser edge network (stub) than "ts5k-small". "ts5k-large" is used to represent a situation in which a system consists of nodes from several big stub domains, while "ts5k-small" represents a situation



(a) Num. of information pieces.



(b) Average outlinks.



(c) Information distribution balance.

Fig. 8. Overhead of different resource discovery methods.

Table 1
Simulated environment and parameters

Parameter	Default value
Object arrival location	Uniform over ID space
Number of nodes	2048
Number of keywords	20,591
Number of resources	10,000
Number of queries	100
L	5
k	2
Distance threshold *r*	3

in which a system consists of nodes scattered in the entire Internet and only few nodes from the same edge network join in the system.

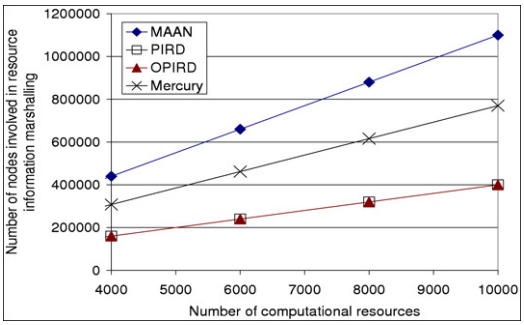
Our testing data set is obtained from Acxiom Corporation which is one of the major providers of integrated customer information in the world. Table 1 lists the parameters of the simulation and their default values for all methods, unless otherwise specified.

4.1. Overhead of resource discovery methods

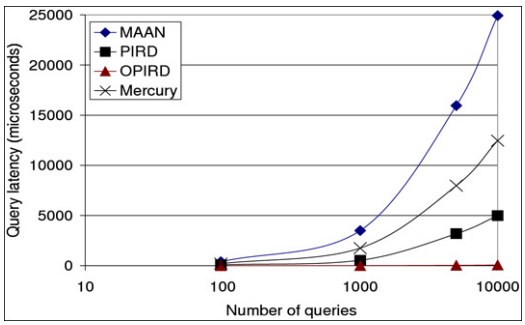
Fig. 8(a) shows the total number of information pieces stored in the system for all available resources. We can see that PIRD and OPIRD generate the same number of information pieces, and their results are much lower than Mercury. In addition, Mercury generates lower number than MAAN. Recall that PIRD and OPIRD change each resource description to L hash values regardless of the number of the keywords in the resource description. Therefore, each resource needs L messages for resource reporting, and there will be L pieces of resource information in the system. Rather than regarding a whole resource description with multiple attributes as an entity, MAAN hashed each keyword in the resource, and stores the resource information in a node responsible for the hashed value. As a result, for a resource description having m keywords, MAAN needs m messages to store m pieces of information. Some resource attributes are described by attribute values such as 1.2 GHz for CPU. Mercury groups attribute name and corresponding value and takes attribute name as keyword to report resource. Therefore, it leads to fewer information pieces than MAAN. Since the average attribute names are greater than L in the experiment, Mercury needs more storage space for the resource information. The results are in agreement with Theorem 3.2 and the Corollary of Theorem 3.2.

Structured P2P overlay maintenance cost constitutes a large part of the overhead of resource discovery mechanisms. In a structured P2P overlay, every node needs to maintain a number of outlinks to connect with its neighbors. Fig. 8(b) plots the average outlinks maintained by a node in different resource discovery approaches. The results show that each node in Mercury maintains dramatically more outlinks than in others. Recall that Mercury has multiple P2P overlays with one P2P overlay responsible for each resource attribute name, such that a node has a routing table for each P2P overlay, and it has a total number of outlinks equal to the product of routing table size and the number of P2P overlays. The results are consistent with Theorem 3.3 in that PIRD/OPIRD can save the outlink maintenance overhead by no less than a factor of m.

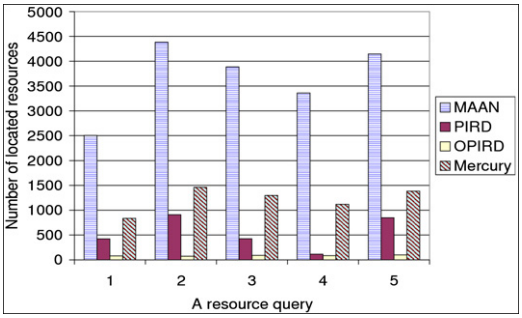
Fig. 8(c) plots the average and the 1st and 99th percentiles of the number of information pieces per node versus the total number of resource information pieces in the system. Two observations can be made from the figure. First, the average size of MAAN is much higher than others due to the same reason observed in Fig. 8(a). Second, MAAN exhibits significantly larger variance than Mercury and PIRD/OPIRD. MAAN maps resource information to a flat structured P2P overlay. Some attribute names appear very frequently such as CPU and Memory, while others are infrequently used such as file name, leading to much more resource information stored in some nodes while only a few stored in others. On the other hand, Mercury uses one structured P2P overlay for each resource attribute, and classifies resource information based on attribute value in each structured P2P overlay. The widespread value ranges help to distribute resource information evenly. Taking advantage of the hierarchical structure of Cycloid, PIRD/OPIRD lets different clusters be responsible for resource information and allocates information to nodes within a cluster based on node



(a) Num. of nodes for marshalling.



(b) Total query time.



(c) Num. of discovered resources.

Fig. 9. Efficiency of different resource discovery methods.

geographical closeness. Therefore, Mercury and PIRD/OPIRD can achieve more balanced distribution of load caused by resource information maintenance and resource discovery operation. The results are in agreement with Theorem 3.4. Considering that Mercury has dramatically high architecture maintenance cost, PIRD/OPIRD should be the most effective with regards to both maintenance overhead and load distribution.

4.2. Efficiency of resource discovery methods

Fig. 9(a) plots the routing nodes involved in resource marshalling. More reporting messages results in more nodes involved for message routing. Thus, we can make the same observations from the experiment results as in Fig. 8(a) due to the same reasons. The results confirm that PIRD and OPIRD need significantly less overhead for resource information polling. This result is consistent with Theorem 3.5.

Fig. 9(b) depicts the query latency of different resource discovery methods versus the number of queries. The figure shows that MAAN generates the highest query latency, followed by Mercury, then PIRD and OPIRD. For a query consisting of m keywords, MAAN generates m queries for all keywords. For example, for a query of workstations described by "Memory 2048 MB CPU 3 GHz", MAAN sends a request for each keyword, collects the resource information for each keyword, and then merges the information to find workstations with indicated features. However, a tremendously high volume of information will be collected, therefore MAAN needs a very long time to prune useless information in the merging phase. Mercury takes both of attribute name and value to request for resource information. Therefore, the information located satisfies both the specified attribute name and values. Thus, Mercury does not need a long time for the final pruning. PIRD sends L requests regardless of the number of keywords in a query. In addition, its refinement further removes the information for resources not satisfying the query. As a result, PIRD improves the query efficiency significantly. Incorporating LZW compression algorithm, OPIRD has much shorter vectors and hence shorter latency for LSH processing, leading to dramatic query latency reduction.

Experiment results show that all discovery methods can return the right results. An effective method should return fewer false positives. Fig. 9(c) shows the number of returned results. We can see that MAAN generates the highest number of turned results, and Mercury leads to more results than PIRD. Since MAAN splits attributes of a resource for resource information collection and query, it returns tremendously high volume of information. On the other hand, Mercury combines resource attribute name and value, so it returned relatively less results. PIRD groups resource information based on their similarity and maps each group to a node in a P2P node, such that it has much fewer false positive results, which means most of its returned results are the information of requested resource.

OPIRD reduces the number of returned resources of PIRD significantly due to its feature of compressed vectors. Because of large dimension and resource vector sparsity, PIRD may locate some resources which do not have common keywords with the query, generating false positives. With appropriate locality-preserving compression, OPIRD can greatly improve the effectiveness of PIRD by eliminating the false positives in its located resource set.

Fig. 10 plots the memory size needed for resource vectors in PIRD and OPIRD. It shows that OPIRD has significantly less memory requirement for vector storage. This confirms the effectiveness of integration of LZW compression algorithm into PIRD for memory reduction. With LZW, the lengths of resource vectors are reduced, hence the memory needed to store resource vectors is reduced greatly.

4.3. Proximity-aware resource discovery

This experiment shows the effectiveness of PIRD/OPIRD in proximity-aware resource discovery, in which resources geographically close to requester nodes are located. In the experiment, we randomly generated 5000 resource requests. Fig. 11(a) and (b) show the Cumulative Distribution Function (CDF) of the

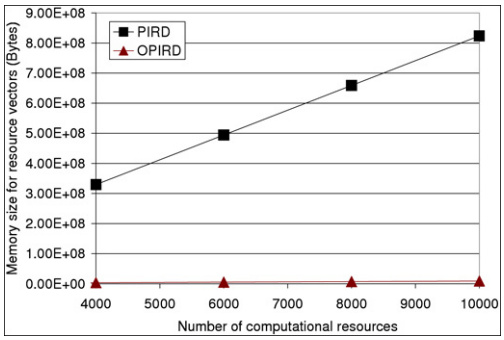
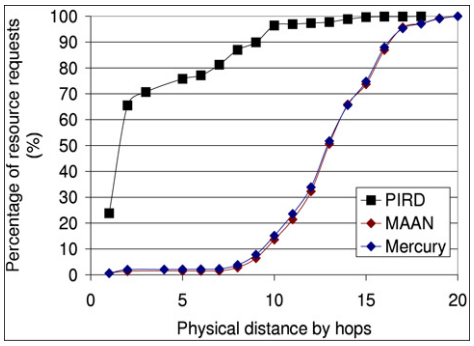
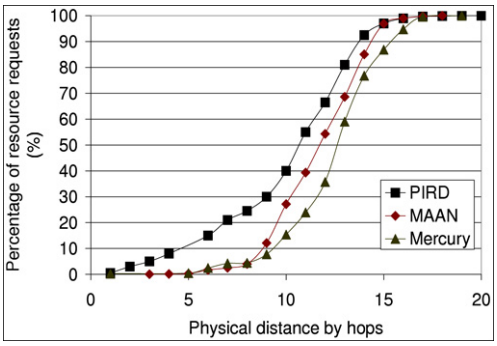


Fig. 10. Memory for resource vectors.



(a) ts5k-large.



(b) ts5k-small.

Fig. 11. Proximity-aware performance.

percentage of resource requests versus the physical distances between resource requesters and providers in "ts5k-large" and "ts5k-small", respectively. In the figure, PIRD also represents OPIRD due to their similar performance. We can see that in "ts5k-large", PIRD is able to locate 97% of total resource requested within 11 hops, while others locate only about 15% within 10 hops. Almost all allocated resources are found within 15 hops from requesters in PIRD, while 19 hops in others. The results show that PIRD can locate most resources within short distances from requesters, while others locate most resource in long distances. From Fig. 11(b), we can make the same observations as in "ts5k-large", although the performance difference between approaches is not so significant. The more resources are located in shorter distances, the higher proximity-aware performance of a resource discovery method. The results indicate that the performance of PIRD mechanism is better than Mercury/MAAN in terms of discovering resources physically close to resource requesters.

5. Conclusions

A key hurdle that must be overcome in resource sharing in the Internet-based distributed computing is efficient resource discovery. In spite of the efforts to develop resource discovery methods, most of them lead to low efficiency and high overhead, while others are not effective in an environment with tremendous number of different resources. In addition, few works can locate resources geographically close to requesters, which is critical to overall system performance. This paper presents a P2P-based intelligent resource discovery mechanism (PIRD) for Internet-based distributed systems. Rather than splitting resource attributes in a resource description, PIRD regards the description of a resource as a whole entity. It relies on locality-preserving LSH hash function and LZW compression function to cluster information of resources with similar attributes together to facilitate efficient resource discovery with low overhead and short query latency. In addition, depending on a hierarchical P2P overlay structure, it further clusters the information of geographically close resources to support geographically proximity-aware resource discovery. Theoretical analysis and experiment results demonstrate the efficiency and effectiveness of PIRD in comparison with other approaches. It dramatically reduces overhead and yields significant improvements in efficiency, and provides high guarantees for resource discovery. Its object-oriented feature, low overhead and high efficiency are particularly attractive to the deployment of Internet-based distributed systems.

Acknowledgments

This research was supported in part by the Acxiom Corporation. We would like to thank the anonymous reviewers for their helpful comments on the draft of this paper. We also thank Ze Li, Ting Li and Yingwu Zhu for their help on the work of LSH and LZW. An early version of this work [35] was presented in the Proceedings of ICDCS'08.

Appendix. Proofs of the effectiveness of LZW

Lemma A.1. *PIRD has higher efficiency on shorter vectors than on longer vectors.*

Proof. In PIRD, the distance between a located resource vector and a query vector is calculated by $d(v,q) = \|v-q\| = \sqrt{\sum_{i=1}^{n}(v_i-q_i)^2}$. During the distance calculation process, once $d > r$, the process is stopped and this located resource is discarded. Therefore, the distance calculation is equivalent to finding how many "1"s in the difference of two vectors. We use p to denote the distribution density of "1"s in a vector. Assume $r = \sqrt{3}$, and there are ten "1"s in the difference of two vectors with length of 20 000, then $p = 0.0005$. The probability that there are three "1"s in the first 300 characters is $P(X = 300) = C_{300}^{3} p^3 (1-p)^{297} \approx 4.5 * 10^{-4}$, which is very small. The probability that three "1"s belonging to the first 300 positions decreases as p decreases. If $P(X = n) = C_n^3 * 10^{-10} \approx 1$, $n \approx 4000$. That is, to identify a substring contain three "1"s with 100% probability, 4000 bits should be scanned. The length of scanned bits increases as p decreases. The time complexity and space complexity of the vectors are $O(l)$, where l is the length of a vector. Both complexities of shorter vectors are smaller than those of longer vectors. ■

Lemma A.2. *The similarity between two strings remains the same before and after the compression using the LZW algorithm.*

Proof. The basic function of the LZW algorithm is to replace the longest repetitive substring with a code in the string translation table. Therefore, each substring corresponds to a code in the translation table. If two original strings have the same substrings, the substrings are compressed into identical codes, maintaining the similarity between the original strings. ■

Lemma A.3. *The LZW algorithm has a higher compression rate for a string that contains a larger number of repetitive substrings, and thus leads to a high compression rate for long resource vectors with repetitive "0"s and "1"s.*

Proof. If a string has many repetitive substrings, all identical substrings can be replaced by one code. If the prefix of the substring has length of l, and is mapped to a code with length of 1, the compression rate of the substring is $l/1$. The compression rate increases as l increases. Consider a string with m repetitive "0"s. We use code i for (00), code $(i+1)$ for (000), code $(i+2)$ for (0000), . . . , code $(i+n-2)$ for string (000 . . . 00000) which has n "0"s, therefore
$1+2+3+\cdots n = m \Rightarrow (1+n)n/2 = m \Rightarrow n \approx \sqrt{2m}$.

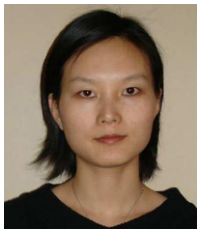
If $m = 20\,000$, then $n = 200$. Therefore, the compression rate is 200. If a resource has only a few dimension attributes in a tremendously high dimension space, it will have a vector with a very large number of "0"s. Using LZW to compress these repetitive "0"s will lead to a high compression rate. ■

Lemma A.4. *The performance of PIRD is improved after the resource and query vectors are compressed by the LZW algorithm.*

Proof. LZW takes 1.5 cycles per input character and per output code [41]. Therefore, compressing a string with length of n takes $1.5n$ cycles. Suppose the compression rate of LZW on sparse resource vector is 100, which means the length of the compressed string is $0.01 * n$. Normally, the cycles for each corresponding character during distance calculation is at least 4 cycles [39]. Therefore, the time saved on the distance calculation of the original string is at least $4n(1-0.01) = 3.96n$. Consequently, the total time saving of the whole process is $3.96n - 1.5n = 2.46n$ cycles, where $1.5n$ is the LZW compression time in cycles. ■

References

[1] A. Andrzejak, Z. Xu, Scalable, efficient range queries for grid information services, in: Proc. of P2P, 2002, pp. 33–40.
[2] T. Asano, D. Ranjan, T. Roos, E. Welzl, P. Widmaier, Space filling curves and their use in geometric data structure, Theoretical Computer Science 181 (1) (1997) 3–15.
[3] Average simultaneous global P2P users. http://www.slyck.com/misc/p2p_history_sep-06_average.xls.
[4] F. Berman, et al., Adaptive computing on the grid using AppLeS, TPDS 14 (4) (2003).
[5] A.R. Bharambe, M. Agrawal, S. Seshan, Mercury: Supporting scalable multi-attribute range queries, in: Proc. of ACM SIGCOMM, 2004, pp. 353–366.
[6] Bittorrent. http://en.wikipedia.org/wiki/Bittorrent.
[7] M. Cai, M. Frank, P. Szekely, MAAN: A multi-attribute addressable network for grid information services, Grid Computing 2 (1) (2004) 3–14.
[8] M. Cai, K. Hwang, Distributed aggregation algorithms with load-balancing for scalable grid resource monitoring, in: Proc. of IPDPS, 2007.
[9] Yan Chen, Randy Katz, On the placement of network monitoring sites, http://www.cs.berkeley.edu/yanchen/wnms, 2001.
[10] A. Chien, B. Calder, S. Elbert, K. Bhatia, Entropia: Architecture and performance of an enterprise desktop grid system, JPDC 63 (5) (2003).
[11] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: Proc. of the 10th IEEE International Symposium on High Performance Distributed Computing, HPDC, 2001.
[12] Z. Drmac, M.W. Berry, E.R. Jessup, Matrices, vector spaces, and information retrieval, SIAM Review 2 (1999) 335–362.
[13] T.S. Eugene Ng, H. Zhang, Towards global network positioning, in: Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurement, 2001.
[14] I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit, Int. J. High Performance Computing Applications 2 (1997) 115–128.
[15] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke, Condor-G: A computation management agent for multiinstitutional grids, in: Proc. IEEE HPDC, 2001.
[16] A. Fu, P.M.S. Chan, Y.L. Cheung, Y.S. Moon, Dynamic VP-tree indexing for n-nearest neighbor search given pair-wise distances, VLDB Journal 2 (2000) 154–173.
[17] C. Germain, V. Neri, G. Fedak, F. Cappello, XtremWeb: Building an experimental platform for global computing, in: Proc. of IEEE/ACM Grid, December 2000.
[18] A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, The VLDB Journal (1999) 518–529.
[19] P. Indyk, R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, in: Proc. of ACM STOC, 1998, pp. 604–613.
[20] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, L. Zhang, On the placement of Internet instrumentation, in: Proc. of INFOCOM, 2000.
[21] D. Karger, E. Lehman, et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web, in: Proc. of STOC, 1997, pp. 654–663.
[22] Lempel–Ziv–Welch Algorithm, http://en.wikipedia.org/wiki/LZW.
[23] LSH Algorithm and Implementation (E2LSH). website http://web.mit.edu/andoni/www/LSH/index.html.
[24] G.S. Manku, M. Bawa, P. Raghavan, Symphony: Distributed hashing in a small wold, in: Proc. of the 4th USENIX Symposium on Internet Technologies and Systems, USITS, 2003.
[25] P. Maymounkov, D. Mazires, Kademlia: A peer-to-peer information systems based on the XOR metric, in: Proc. of IPTPS, 2002.
[26] M. Mutka, M. Livny, Scheduling remote processing capacity in a workstation-processing bank computing system, in: Proc. of ICDCS, September 1987.
[27] M.O. Neary, S.P. Brydon, P. Kmiec, S. Rollins, P. Capello, Javelin++: Scalability issues in global computing, Future Generation Computing Systems Journal 15 (5–6) (1999) 659–674.
[28] D. Oppenheimer, J. Albrecht, D. Patterson, A. Vahdat, Scalable wide-area resource discovery, Technical Report TR CSD04-1334, Univ. of California, 2004.
[29] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proc. of ACM SIGCOMM, 2001, pp. 329–350.
[30] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, Topologically-aware overlay construction and server selection, in: Proc. of INFOCOM, 2002.
[31] S. Ratnasamy, J.M. Hellerstein, S. Shenker, Range queries over DHTs. Technical Report IRB-TR-03-009, Intel Corporation, 2003.
[32] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz, Handling Churn in a DHT, in: Proc. of the USENIX Annual Technical Conference, 2004.
[33] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, in: Proc. of Middleware, 2001, pp. 329–350.
[34] C. Schmidt, M. Parashar, Flexible information discovery in decentralized distributed systems, in: Proc. of HPDC, 2003, pp. 226–235.
[35] H. Shen, Z. Li, T. Li, PIRD: P2P-based intelligent resource discovery in Internet-based distributed systems, in: Proc. of the 28th International Conference on Distributed Computing Systems, ICDCS, 2008.
[36] H. Shen, C. Xu, G. Chen, Cycloid: A scalable constant-degree P2P overlay network, Performance Evaluation 63 (3) (2006) 195–216.
[37] H. Shen, C.-Z. Xu, Hash-based proximity clustering for efficient load balancing in heterogeneous DHT networks, Journal of Parallel and Distributed Computing (JPDC) (2008).
[38] I. Stoica, R. Morris, D. Liben-Nowell, et al., Chord: A scalable peer-to-peer lookup protocol for internet applications, TON 1 (1) (2003) 17–32.
[39] J. Suh, D. Kang, S.P. Crago, Efficient Algorithms for Fixed-Point Arithmetic Operations in an Embedded PIM. SCI, 2001.
[40] D. Talia, P. Trunfio, J. Zeng, M. Högqvist, A DHT-based Peer-to-Peer framework for resource discovery in grids. Technical Report TR-0048, Univ. of California, 2006.
[41] Terry A. Welch, A technique for high performance data compression, IEEE Computer 6 (1984) 8–19.
[42] Z. Xu, et al. Turning heterogeneity into an advantage in overlay routing, in: Proc. of INFOCOM, 2003.
[43] E. Zegura, K. Calvert, S. Bhattacharjee, How to model an internetwork, in: Proc. of INFOCOM, 1996.
[44] B.Y. Zhao, L. Huang, et al., Tapestry: An infrastructure for fault-tolerant wide-area location and routing, J-SAC 12 (1) (2004) 41–53.
[45] Y. Zhu, Y. Hu, Efficient semantic search over DHT overlays, Journal of Parallel and Distributed Computing (JPDC) (2007).



Haiying Shen received the B.S. degree in Computer Science and Engineering from Tongji University, China in 2000, and the M.S. and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Assistant Professor in the Department of Computer Science and Computer Engineering of University of Arkansas. Her research interests include distributed and parallel computer systems and computer networks, with an emphasis on peer-to-peer and content delivery networks, wireless networks, and resource management in cluster and grid computing. Her research work has published in top journals and conferences in these areas. She was a co-Chair of the RFID track in IEEE-CASE 2008, a PC member of many conferences such as ICPP and EUC. She is a member of IEEE and ACM.