# PIRD: P2P-based Intelligent Resource Discovery in Internet-based Distributed Systems

Haiying Shen[1], Ze Li[3], Ting Li[4]
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, AR 72701
{hshen, zxl008, txl005}@uark.edu

Yingwu Zhu[2]
Dept. of Computer Science and Software Engineering
Seattle University, Seattle, WA 98122
zhuy@seattleu.edu

## 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 each message for an attribute, leading to high overhead. Anther approach can reduce multi-attribute to one 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. It further incorporates Lempel-Ziv-Welch algorithm to compress attribute information for higher efficiency. In addition, it helps to search resources geographically close to requesters by relying on a hierarchical P2P structure. PIRD significantly reduces overhead and improves the efficiency and effectiveness of resource discovery. 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. 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. Multi-attribute resource discovery refers to the problem of locating resources that are described by a set of attributes (e.g., OS version, CPU speed, etc.). Efficient multi-attribute resource discovery technology is indispensable for resource sharing in Internet-based distributed systems.

A fundamental problem in these large, decentralized, distributed resource sharing environments is efficient discovery of resources in the absence of global knowledge of naming conventions. Another challenge comes from the complex environment characterized by large scale, geographically scattered, and dynamism. 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. The third challenge is to offer guarantees and support flexible searches using keywords, wildcards, and range queries.

Centralized resource discovery approaches [8, 6, 7] 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) middleware [23, 17, 22] for resource discovery [2, 4, 5, 19] 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 information marshalling. For a resource "CPU=2GHz and Memory=512MB," one group of methods [2, 4] use multiple P2Ps with each P2P responsible for one attribute name such as CPU and memory, and use two messages using the attribute values as keywords to store the information in two P2Ps, respectively. However, multiple P2P structures require high maintenance overhead, especially in dynamism.

1063-6927/08 $25.00 © 2008 IEEE
DOI 10.1109/ICDCS.2008.9

Another group [5] takes both attribute name and value as keywords, and use four messages to store the resource information in a single P2P. However, it will lead to imbalance of information and load distribution. 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. They lead to high overhead for information storing, reporting/searching and subsequently merging operation. A resource with $m$ keywords in the description needs $m$ messages for resource marshaling and searching. They need to merge a tremendously high volume of discovered information to derive the information of desired resources. To avoid attribute splitting, another class of approaches [18] 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 are critical to system performance.

This paper proposes a P2P-based intelligent information searching (PIRD) mechanism that weaves all attributes into a set of indices and maps the indices to one P2P using locality sensitive hashing (LSH) [13, 12]. PIRD is object-oriented in that it regards the description of a resource such as a computer and a file as an 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, and improves resource discovery efficiency. PIRD further uses Lempel-Ziv-Welch (LZW) algorithm to compress attribute information to reduce overhead and improve efficiency. In addition, by taking advantage of the hierarchical structure of a structured P2P, it helps to find resources geographically close to requesters.

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 introduction, LSH and LZW algorithms. Section 4 shows the performance of PIRD using a variety of metrics. Section 5 concludes this paper.

## 2 Related Work

There have been numerous resource discovery approaches to discover resources in Internet-based distributed systems. Systems such as Condor-G [9] uses the Globus toolkit [8] to integrate with a grid computing environment for resource management. A number of projects, including Condor [14], XtremWeb [11], Entropia [6], AppLes [7], and Javelin++ [15], have investigated resource searching for computations on grid systems. However, relying on centralized or hierarchical based policies, these systems have limitation to explore 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.

To achieve multi-attribute resource discovery, some systems adopt one structured P2P for each attribute, and process multi-attribute queries in parallel in corresponding P2Ps [2, 4]. Depending on multiple P2Ps for multi-attribute resource discovery leads to high maintenance overhead for P2P structures. Another group of approaches [5, 19] organizes all resource information into one structured P2P overlay and lets a node be responsible for all information of resources with the same attribute. This approach results in load imbalance among nodes, and leads to high cost for searching resource information among a huge volume of information in a single node. In all 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 a $m$-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. Schmidt and Parashar [18] proposed a dimension reducing indexing scheme that maps the multidimensional information space to P2P nodes. Though it 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 dimension reduction algorithm in a high-dimensional space. More importantly, most current P2P-based methods are unable to discover resources geographically close to requester, which is very important for resource sharing performance.

Unlike the three groups of approaches, the PIRD mechanism achieves balanced load distribution with low P2P structure maintenance overhead. Furthermore, it can search 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 structured P2P overlay network to achieve multi-attribute resource discovery. Before we begin more detailed discussion of PIRD,

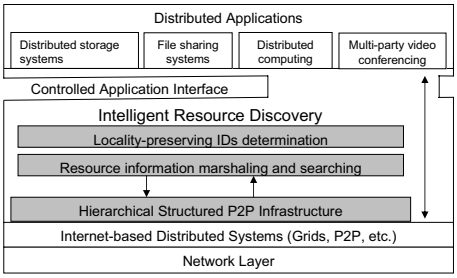



**Figure 1. High level architecture of PIRD.**

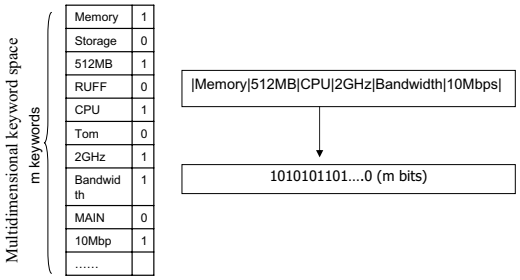



**Figure 2. Resource vector generation.**

we briefly describe a hierarchical structured P2P overlay network called Cycloid followed by a high-level view of PIRD architecture and its components. Cycloid [22] is a lookup efficient constant-degree overlay. It has $n{=}d \cdot 2^d$ nodes and $d$ is its dimension. Each Cycloid node is represented by a pair of indices $(k, a_{d-1}a_{d-2}\ldots a_0)$, where $k$ is a cyclic index and $a_{d-1}a_{d-2}.....a_0$ is a cubical index. The cyclic index is an integer, ranging from 0 to $d-1$ and the cubical index is a binary number between 0 and $2^d-1$. The nodes with the same cubical index are ordered by their cyclic index mod $d$ on a small cycle, which we call *cluster*. All clusters are ordered by their cubical index mod $2^d$ on a large cycle. The right part of Figure 4 shows the partial routing links of a 11-dimensional Cycloid, where $x$ indicates all possible cyclic index. Cycloid provides two mains functions: `Insert(key,object)`, `Lookup(key)` to store an object to a node responsible for the key, and to retrieve the object. For more information about Cycloid, please refer to [22].

Figure 1 shows a high level architecture of PIRD. It relies on a single Cycloid with low maintenance overhead. PIRD includes two characteristic components to effectively and effectively search resources for high performance Internet-based distributed systems.

**Locality-preserving ID determination.** The fundamental functionality of this component is to represent each resource and query by a set of P2P IDs that preserve locality. Typically, a resource/query is described by a vector where each dimension is associated with a distinct keyword that represents a resource attribute. The vector is then used to produce a small set of IDs through LSH as the P2P IDs for the resource. Consequently, resources/queries with similar vectors will have similar IDs.

**Resource information marshalling and searching.** This component provides resource registering and retrieval capabilities. The functionality of reporting is to store the information of each resource automatically to a structured P2P according to resource keyword vector. The functionality of resource searching is to locate desired resources for a given query. P2P lookup function facilitates requesters to discover resources efficiently, and hierarchical Cycloid P2P further enables requesters to locate geographically close resources for high performance.

### 3.1 Locality-preserving ID Determination

A key component of a resource discovery system is defining an key space and deterministically mapping resources to this key space. To support complex resource searches in a resource discovery system, we associate each resource with a sequence of keywords and define a mapping that preserves keyword similarity.

The keywords are common words to describe resource attributes such as bandwidth and memory, and values of globally defined resource attributes such as 10Mbps and 512MB. As a result, each resource is represented by a keyword vector. All keywords form a multidimensional keyword space where resources are points in the space and the keywords are the coordinates. Two resources are considered "local" if they are close together in this keyword space. For example, "Memory 512MB CPU 2GHz" and "Memory 1GB CPU 2GHz" are local as they have many common keywords. Figure 2 demonstrates how a resource's vector is determined, and Figure 3 shows an example of a keyword space.

The problem of resource discovery can be regarded as finding the nearest neighbor of a query point in a high dimensional keyword space focusing mainly on the Euclidean space: given $n$ points in a $m$ dimensional space, find the nearest neighbor of a query point. The next question is how to transform resource vectors to IDs in an index space. Hilbert space-filling curve (SFC) [3] and locality sensitive hashing (LSH) [12] are two main technologies for dimension reduction while still preserving the relative distances among points in a multi-dimensional space.

**Hilbert space-filling curve.** SFC maps points in a 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. The 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. We call this number the *Hilbert number* of a node. The Hilbert number indicates physical closeness of nodes on the Internet.

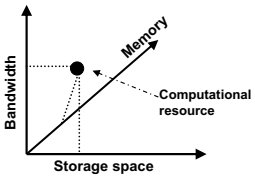



**Figure 3. A 3-D keyword space.**

**Locality Sensitive Hashing.** 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 bucket 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 bucket 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 $B(q, r)$ represents the scope with range $r$ around point $q$, and $\mathbf{Pr}$ denotes probability.

**Definition** A family $\mathcal{H} = \{h : S \rightarrow U\}$ is called $(r_1, r_2, p_1, p_2)$-sensitive 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 $\mathbf{Pr}_{\mathcal{H}}[h(q) = h(v)] \leq p_2$,
–$r1 < r2, p1 > p2$.

Schmidt and Parashar [18] used SFC to construct a key space, and proved the effectiveness of the SFC adoption for resource discovery in 3-dimensional space. They indicated that SFC's effectiveness will be degraded with increasing dimension. In real practice, there are tremendously different resources including data resources such as files, videos and audios, computing resources such as CPU time, storage, and other resources such as expertise and devices. Therefore, the number of keywords used to describe millions of various resources is enormously high. Therefore, SFC cannot be practically applied to resource discovery where there are significantly large number of keywords. Based on this observation, we chose LSH for resource discovery in a high-dimensional space.

**LSH-based resource ID determination.** In the following, we discuss how resource information and queries are mapped into the underlying structured P2P according to their keywords using LSH. The goal of mapping is to cluster the information of similar resources to the same nodes with high probability.

Different LSH families can be used for different distance functions. PIRD relies on the LSH technique in Euclidean spaces proposed by Datar *et al.* [10] that uses p-stable distributions.

Based on the distribution, a family $\mathcal{H}$ of hash functions is derived. That is, $h_{a,b}(v) = [\frac{a\dot{v}+b}{w}]$, where $a$ and $b$ are randomly generated value by $g(x)$, and $w$ is a specified value. PIRD defines a function family $G = g : S \rightarrow U^k$ such that

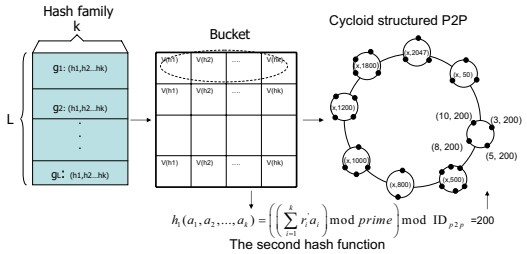



**Figure 4. Process of ID mapping in PIRD.**

$g(v) = (h_1(v), ..., h_k(v))$, where $h_i$ belongs to $\mathcal{H}$, and $k$ is a specified integer value that represents a tradeoff between the time spent in computing hash values and time spent in pruning false positives. False positives are returned results but are not actually close to a query point.

The hashed value of each resource vector $v$, $h_i(v)(1 \leqslant i \leqslant k)$, is stored in bucket $g_j(v)(1 \leqslant j \leqslant L)$, where $L$ is determined by $k$. Then PIRD uses the hash function

$$h_1(a_1, a_2, ..., a_k) = ((\sum_{i=1}^{k} r_i^{'} a_i) mod\ prime) mod\ ID_{p2p},$$

to compute the index of each bucket. In the hash function, $ID_{p2p}$ represents the P2P ID space size, $r_i^{'}$ is a random number, and $prime$ denotes a prime number. The indices of a record $v$ are their final IDs for mapping to a P2P. Figure 4 illustrates ID mapping to P2P in PIRD. In conclusion, given a vector $v$, PIRD hashes $v$ to $L \times k$ hash values, and gets the final $L$ hash values of each bucket. Finally, similar resources/vectors have similar IDs, and their information are stored in the same node or close nodes in P2P.

To search near neighbors, a distance is initially set to a specified value $r$. To process a query $q$, PIRD also makes buckets $g_1(q), ..., g_L(q)$ first; then searches neighbors based on the hashed values of $g_1(q), ..., g_L(q)$ in the P2P. Let $v_1, ..., v_t$ be the resource vectors found for $q$, PIRD then computes the Euclidean space distance between $q$ and $v_1, ..., v_t$ using $d(x, y) = ||x - y|| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. For each $v_i$ if $v_i$ belongs to $B(q, r)$, which means $d(x, y) \leq r$, then $v_i$ is a similar point in range $r$. Otherwise, $v_i$ is not the point satisfying to query $q$. This refinement is to prune false positive results.

### 3.2 Resource Information Marshalling and Searching

After the IDs of a resource in P2P ID space are determined, the next question is how to map information resource information to a structured P2P for efficient searching. Recall that a P2P overlay network provides two main functions: `Insert(key,object)` and `Lookup(key)`. By having IDs as the P2P keys, a resource is indexed into the structured P2P in the form of $< ID, (v, ip\_addr) >$. For instance, a resource of "Memory 512MB CPU 2GHz Bandwidth 10Mbps" gets $L$ IDs,

$$ID_1, ID_2, ..., ID_L$$

using the LSH-based locality-sensitive ID generation method. It then uses

$< ID_1, (v, ip\_addr) >, ..., < ID_L, (v, ip\_addr) >$

to insert its resource information to the structured P2P. Consequently, its information is stored in $L$ nodes in the P2P. Note that for two vectors $v_1$ and $v_2$, their similarity is

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

Such resource information collection can have the IDs of similar resources hashed to the same nodes with $p \geq 1 - (1 - p^m)^n$. The resource information marshalling may incur load imbalance. Load balancing algorithm [21] can be adopted into PIRD for load balance.

We now discuss the issue of how to locate resources that satisfy a query, given the fact that all resources in the system are automatically indexed according to their vectors of their keywords. Processing a query consists of two steps: translating the keyword query to relevant P2P IDs, and querying the appropriate nodes in the overlay network for resource information. After the P2P IDs associated with a query are identified, straightforward query processing consists of sending a query message for each P2P ID. A query message targeting each ID is routed to the appropriate node based on structured P2P lookup algorithm. 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. Finally, the nodes containing the information of the queried resource are located.

For example, let $v$ be a query Q's keyword vector. Suppose Q wants to locate those resources whose vectors are similar to $v$ (within a certain distance). PIRD produces $L$ IDs from $v$ for Q using the same set of hash functions in the LSH. Therefore, if a resource satisfies query Q, it will be retrieved by Q with very high probability. Note that the vectors of resources and query Q could be hashed to the same IDs with high probability (i.e., $1 - (1 - p^m)^n$). Thus, by having these IDs as the P2P keys in `Lookup(key)`, Q is able to retrieve desired resources from the nodes responsible for these IDs. $L$ 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 $\leq L$. Upon receiving a request, each destination node locally checks the list of tuples $< ID, (v, ip\_addr) >$ and returns the $ip_addr$s of nodes that have resources similar to the query's $v$ with certain similarity threshold $r$. Then, the requester merges the replies from all destination nodes, and asks resources from resource owners with $ip_addr$s.

The following theorems show the performance of PIRD compared with other resource discovery methods. Please see [20] for details of proofs.

**Theorem 3.1** *For a resource with $m$ attributes each of which has $k$ keywords, MAAN [5] and Mercury [4] store the information of the resource to $m \times k$ P2P nodes and $m \leq x \leq m \times k$ nodes, respectively. PIRD stores the location of the computer resource to $\leq L$ nodes.*

**Theorem 3.2** *For a resource with $m$ attributes each of which has $k$ keywords, MAAN and Mercury need $m \times k$ and $m \leq x \leq m \times k$ messages to report/query the resource to/from destination nodes, respectively, while PIRD only needs $\leq L$ messages.*

### 3.3 Proximity-aware Resource Location

Locating geographically close resources is important to the performance of Internet-based distributed applications, especially time-critical applications. We improve PIRD by letting it take into account resource geographical locality.

First, let us introduce a landmarking method to represent node closeness on the Internet by indices. Landmark clustering has been widely adopted to generate proximity information [16, 25]. 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. We assume $m$ 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 $< d_1, d_2, \ldots, d_m >$ as its coordinate in Cartesian space. Two physically close nodes will have similar landmark vectors. We use SFC to map $m$-dimensional landmark vectors to Hilbert numbers, such that the closeness relationship among the points is preserved. We use $\mathbb{H}$ to denote the Hilbert number of a node.

Recall that Cycloid consists of a number of clusters, which constitute a large cycle. We let each cluster be responsible for the information of similar resources, and divide the information to nodes within the cluster based on resource geographically closeness. In a Cycloid ID, the cubical indices differentiate clusters, while the cyclic indices indicate different node positions in a cluster. We use cubical indices to represent different resources, and use cyclic indices to represent the location of nodes. We let nodes report their load information to the system by `Insert`$((\mathbb{H}, ID), (v, ip\_addr))$. Based on key assignment algorithm, the information of similar resources will be in a same cluster. Within each cluster, the information of resources in close proximity will gather together in a node. When a node queries for different resources, it only needs to send out different requests using `Lookup` $(\mathbb{H}, ID)$ for each resource ID. Each request for a resource will be forwarded to the node responsible for the information of the resource in geographically close proximity. For a flat structured P2P, when a node reports the information of its resources, it also reports its Hilbert number $\mathbb{H}$. Each destination node organizes its own tuples in such a way that these tuples are clustered locally based on their owner's Hilbert number. When a node $i$ queries for a resource, it also sends its Hilbert number $\mathbb{H}_i$ along with the request. When a destination node receives the query, it only needs to check the cluster whose $\mathbb{H} \approx \mathbb{H}_i$. Consequently, node $i$ receives the information of resources that locate geographically close to

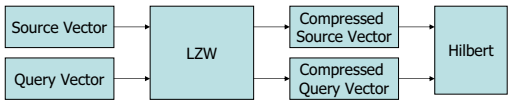



**Figure 5. Integration of LZW into LSH.**

**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 |

it. Communicating with geographically close nodes, and using geographically close resources improves the performance of applications significantly.

### 3.4 Optimized PIRD

In an Internet-based distributed systems 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 by LSH. For instance, a Q only wants memory and CPU in a 10,000-dimensional keyword space, then its vector will have two 1 bits with all other bits equal to 0. LSH needs to be complemented by a compress algorithm to make it more efficient. This is confirmed by Lemma 3.1. Due to the space limit, please see [20] for proofs of the Lemmas.

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

To optimize LSH in processing long but sparse vectors, we adopt LZW dynamic compression algorithm [24] to reduce the dimension of vectors and remove insignificant strings, i.e. 0s. LZW is a universal lossless data compression algorithm. It replaces strings of characters with single codes. After compression, the new string still keeps the old string's information. For example, for a string ABCCAABCDDAACCDB, if LZW uses 4 to denote AB, and 5 to denote CC, after compression, the string becomes 45A4CDDAA5DB. Figure 5 shows a model of the integration of LZW algorithm to the LSH process.

## 4 Performance Evaluation

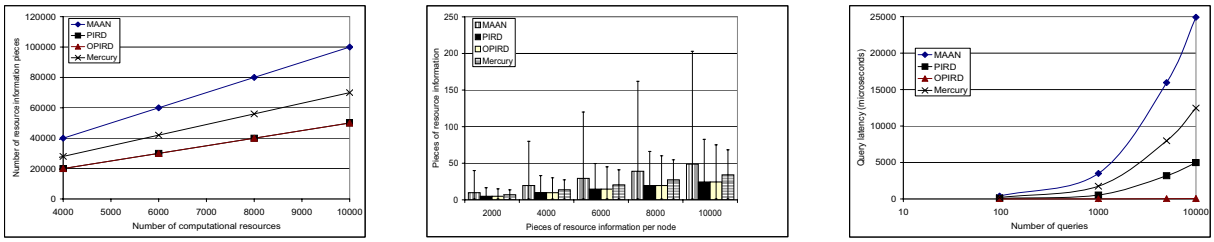
We designed and implemented a simulator in Java for evaluation of PIRD and optimized PIRD (OPIRD) based on Cycloid hierarchical P2P [22] and E2LSH 0.1 [1]. We compared the performance of PIRD with MAAN [5] and Mercury [4] on Chord [23], in terms of storage requirement for resource information marshaling, load balance, resource search time, and proximity-aware performance. We conducted an experiment on SFC [18], and found that SFC is not effective in a large dimension space. All records are hashed to the same value. This is consistent to the statement in [18] that the SFC's performance degrades when the number of dimensions increases to a large number.

We used two transit-stub topologies generated by GT-ITM [26]: "ts5k-large" and "ts5k-small". "ts5k-large" is used to represent a situation in which a P2P system consists of nodes from several big stub domains, while "ts5k-small" represents a situation in which a P2P system consists of nodes scattered in the entire Internet and only few nodes from the same edge network join in the overlay. Table 1 lists the parameters of the simulation and their default values, unless otherwise specified.

### 4.1 Efficiency of Resource Discovery Methods

Figure 6(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 record to $L$ hash values regardless of record length. Therefore, each record needs $L$ messages for resource reporting, and there will be $L$ pieces of information of the resource in the system. Rather than regarding a whole record as an entity, for a $m$ keywords record, MAAN needs $m$ messages to store $m$ pieces of information. 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 number of attribute names is greater than $L$ in the experiment, Mercury needs more storage space for the resource information. The results are in agreement with Theorem 3.1 and Theorem 3.2.

Figure 6(b) plots the average and the 1st and 99th percentiles of pieces of information in per node versus the total piece number of resource information. 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 Figure 6(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 varied load imbalance. On the other hand, Mercury uses one structured P2P for each resource attribute, and classifies resource information based on value in each structured P2P. The widespread value ranges helps to distribute resource information evenly. Taking advantage of the hierarchical structure of Cycloid,




(a) Num. of information pieces (b) Information distribution balance (c) Total query time

**Figure 6. Efficiency of different resource discovery methods.**

PIRD/OPIRD lets different clusters be responsible for resource information in the cluster and allocates information to nodes based on geographically closeness. Therefore, Mercury and PIRD/OPIRD can achieve more balanced distribution of load due to resource information maintenance and resource discovery operation.

Figure 6(c) depicts the query time of different resource discovery methods versus the number of queries. The figure shows that MAAN leads to the highest query latency, followed by Mercury, PIRD and then OPIRD. For a query consisting of $m$ keywords, MAAN generates $m$ queries for all keywords. Hence, it needs a very long time to prune useless information in the information merging phase. Mercury takes both of attribute name and value to request for information. Therefore, the discovered resources have requested attribute name and values. As a result, Mercury does not need long time for 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 to the requesters. PIRD improves the efficiency of query significantly. Incorporating LZW compression algorithm, OPIRD has much shorter IDs and hence shorter time for LSH processing, leading to dramatically query latency reduction.

Experiment results show that all discovery methods can return right results. An effective method should return fewer false positive records. Figure 7(a) shows the number of returned results. We can see that MAAN generates the highest number of turned results, and Mercury incurs more results than PIRD. MAAN splits attributes of a resource for resource information collection and query. Due to its manner of information collection and query as explained above, MAAN returned tremendously high volume of information. On the other hand, Mercury combines resource attribute name and value, so it returned relatively less results. PIRD clusters resource information based on their similarity and maps each cluster 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 records of PIRD significantly due to its compressed IDs feature. Because of large dimension and record ID sparsity, Euclidean Space Distance computation may not be able to avoid false positives. Hence, PIRD generates a large number of false positives. With appropriate locality-preserving compression, OPIRD with compression greatly improves the effectiveness of PIRD by reducing the false positives in its returned record set.

### 4.2 Proximity-aware Resource Discovery

This experiment shows the effectiveness of PIRD/OPIRD in proximity-aware resource discovery, in which resources geographically close to requesters nodes are located. In the experiment, we randomly generated 5000 resource requests. Figure 8(b) and (c) show the CDF of the percentage of resources requests versus distances between resource requesters and providers. 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 located 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 allocate most resource in long distances. From Figure 8(c), 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

In spite of the efforts to develop resource discovery methods in Internet-based distributed systems, most of them lead to low efficiency and high overhead due to splitting resource attributes, while others are not effectiveness in an environment with tremendously different resources. In addition, there have been very little research devoted to tackling the challenge of locating resources geographically close to requesters. This paper presents P2P-based Intelligent Resource Discovery mechanism (PIRD). PIRD regards the de-

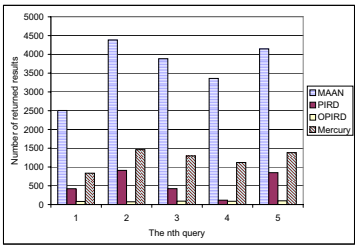
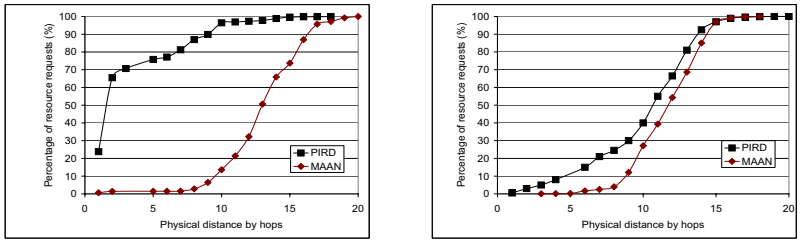



**Figure 7. Num. of discovered resources.**




(a) ts5k-large (b) ts5k-small

**Figure 8. Proximity-aware performance.**

scription as an entire 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. In addition, depending on a hierarchical P2P structure, it further supports geographically proximity-aware resource discovery. It dramatically reduces overhead and yields significant improvements in efficiency. 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 U.S. Acxiom Corporation.

## References

[1] LSH Algorithm and Implementation (E2LSH). website:http://web.mit.edu/andoni/www/LSH/index.html.

[2] A. Andrzejak and Z. Xu. Scalable, Efficient Range Queries for Grid Information Services. In *Proc. of P2P*, pages 33–40, 2002.

[3] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier. Space filling curves and their use in geometric data structure. *Theoretical Computer Science*, 181(1):3–15, 1997.

[4] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proc. of ACM SIGCOMM*, pages 353–366, 2004.

[5] M. Cai, M. Frank, and P. Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. *Grid Computing*, 2(1):3–14, 2004.

[6] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: Architecture and Performance of an Enterprise Eesktop Grid System. *JPDC*, 63(5), May 2003.

[7] F. B. et. al. Adaptive Computing on the Grid Using AppLeS. *TPDS*, 14(4), Apr. 2003.

[8] I. Foster and C. Kesselman. Globus: a Metacomputing Infrastructure Toolkit. *Int. J. High Performance Computing Applications*, 2:115–128, 1997.

[9] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: a Computation Management Agent for Multiinstitutional Grids. In *Proc. IEEE HPDC*, 2001.

[10] 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*, (2):154–173, 2000.

[11] C. Germain, V. Neri, G. Fedak, and F. Cappello. XtremWeb: Building an Experimental Platform for Global Computing. In *Proc. of IEEE/ACM Grid*, Dec. 2000.

[12] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. *The VLDB Journal*, pages 518–529, 1999.

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of ACM STOC*, pages 604–613, 1998.

[14] M. Mutka and M. Livny. Scheduling Remote Processing Capacity in a Workstation-Processing Bank Computing System. In *Proc. of ICDCS*, September 1987.

[15] M. O. Neary, S. P. Brydon, P. Kmiec, S. Rollins, and P. Capello. Javelin++: Scalability Issues in Global Computing. *Future Generation Computing Systems Journal*, 15(5-6):659–674, 1999.

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

[17] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*, pages 329–350, 2001.

[18] C. Schmidt and M. Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *Proc. of HPDC*, pages 226–235, 2003.

[19] 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.

[20] H. Shen, Z. Li, and T. Li. PIRD: P2P-based Intelligent Resource Discovery in Internet-based Distributed. Technical Report TR-08-062, University of Arkansas, 2008.

[21] H. Shen and C. Xu. Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured Peer-to-Peer Networks. *TPDS*, 2007.

[22] H. Shen, C. Xu, and G. Chen. Cycloid: A Scalable Constant-Degree P2P Overlay Network. *Performance Evaluation*, 63(3):195–216, 2006.

[23] I. Stoica, R. Morris, D. Liben-Nowell, and et al. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *TON*, 1(1):17–32, 2003.

[24] T. A. Welch. A Technique for High Performance Data Compression. *IEEE Computer*, (6):8–19, 1984.

[25] Z. Xu and et al. Turning Heterogeneity into an Advantage in Overlay Routing. In *Proc. of INFOCOM*, 2003.

[26] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc. of INFOCOM*, 1996.