Combining Efficiency, Fidelity and Flexibility in Grid Information Services

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

***Abstract*—Grid information service is an inherent component of a grid system which collects resource data and provides search functionality. In addition to efficiency, fidelity in discovering desired resources is also an important metric to evaluate the service quality. Recently, a number of grid information service systems have been proposed based on DHTs. However, these systems either achieve high fidelity at low efficiency, or achieve high efficiency at low fidelity. Moreover, some systems have limited flexibility by only providing exact-matching services or by describing a resource using a pre-defined list of attributes. This paper presents a grid information service, which offers high efficiency and fidelity without restricting resource expressiveness and meanwhile provides similar-matching service. Simulation results show that the proposed service outperforms other services in terms of efficiency, fidelity and flexibility.**

I. Introduction

Over the last few years, grid computing has dramatically evolved from its roots in science and academia, and is currently at the onset of mainstream commercial adoption. Grid computing creates a virtual supercomputer by providing an infrastructure for sharing computing resources between widely-spread computers to maximize the value of computing resources. A resource is always described by a set of attributes such as CPU speed, memory, OS version and device name. For example, if a user needs a computer with resource attributes "OS name=Linux", "CPU speed=1000MHz" and "Free memory=1024MB" for a computing task, how can it quickly discover the required resources? Grid information service is an inherent component of a grid system which collects resource data and provides search functionality in order to bridge resource providers and consumers. However, the problems faced by a grid information service is formidable due to three essential challenges. The first challenge is to achieve high efficiency in grid environment characterized by large scale geographically scattered resources 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 second challenge is to guarantee high fidelity of locating resources. By fidelity, we mean the ability to locate all resources in the system satisfying a resource request. A method with higher fidelity misses less satisfying resources in the system. The third challenge is flexibility which allows users to specify their interested resources with unlimited expressiveness, and to conduct similar resource searching. Similar resources are the resources with similar resource descriptions. Resource descriptions with more common attributes have higher similarity.

Distributed Hash Tables (DHTs) [1–3] provide an attractive solution for grid information services by their inherent properties of self-organization, fault-tolerance, and scalability. However, most of DHT-based services either achieve high fidelity at low efficiency or achieve high efficiency at low fidelity. Moreover, the systems have limited flexibility by only providing exact-matching services or by describing a resource using a pre-defined list of attributes.

One group of the DHT-based systems [4–7] separate attributes of a computer resource, and map the resource into a DHT overlay based on each attribute. We call this group of systems "direct mapping". Since all information of resources containing a specific attribute is stored in one node, the approaches result in load imbalance among nodes, and lead to high cost for resource searching among a huge volume of information. When a requester searches a resource, it searches each attribute of the resource and then merges the results. Though these approaches have high fidelity, they incur a high overhead for resource pooling, searching and merging operation. The direct mapping systems only provide exact-matching service.

Another system [8] combines all attributes of a resource into a single key, and then maps the resource to the key's owner in a DHT overlay. We call this system "one-point mapping". However, converting a number of attributes into a single key may not accurately preserve the similarity between the sets of attributes, especially when there are a large number of attributes in a resource. One-point mapping systems offer high efficiency but at the cost of low fidelity. Recently, we proposed PIRD [9] resource discovery mechanism that weaves all attributes into a set of indices using locality sensitive hashing [10], and then maps the indices to a DHT overlay. Both PIRD and the one-point mapping system have limited flexibility with restricted expressiveness. Users' requested resource attributes are confined to the pre-defined resource attributes.

This paper presents a Locality sensitive hashing based Grid information Service (LGS) combining efficiency, fidelity and flexibility. LGS offers high efficiency and fidelity without restricting resource expressiveness and meanwhile provides similar-matching service. We propose three algorithms to

transform a resource description to a set of integers. We further build locality sensitive hash functions by combing the algorithms with min-wise independent permutations [11]. The hash functions generate a set of IDs for a resource that preserve the similarity of resources without requiring a predefined attribute list. Based on the generated IDs, LGS stores the information of the resource to a DHT overlay. Relying on the DHT object location protocol, the information of requested resource can be efficiently discovered.

II. Related Work

In the DHT-based direct mapping systems, some systems adopt one structured P2P for each attribute, and process resource queries in parallel in corresponding P2Ps [6, 7]. Thus, if a grid has m resource attributes, the grid information service needs m DHT overlays. Depending on multiple P2Ps for resource discovery leads to high maintenance overhead for P2P structures. Another group of approaches [4–7, 12] organizes all resource information into one structured P2P overlay. In this group, attributes of a resource are separated, and the resource information of each attribute is pooled in a P2P node. This strategy results in load imbalance among nodes, and leads to high cost for searching among a huge volume of information in a single node. In most direct mapping systems, when a requester searches a resource, it searches each attribute of the resource and then merges the results. These methods lead to a large number of messages and routing nodes involved in pooling and searching, and hence high cost for information storage and location. They provide high fidelity but at the cost of low efficiency. In addition, the systems only provide exact-matching service which prevents them from wide adoption in practical applications.

Schmidt and Parashar [8] proposed a dimension reducing indexing scheme for resource discovery. They build a multi-dimensional space with each coordinate representing a resource attribute, and project a multi-attribute resource to a point in the space. The proposed scheme then transforms the multiple dimension of a resource to one-dimension using space-filling curve (SFC) [13], while still preserving the similarity between the resources. The system then maps the resource point to a P2P node. It guarantees that all existing resources that match a query are found with bounded costs in terms of the number of messages and nodes involved. However, the scheme is not effective in discovering satisfying resources when there are a large number of attributes because of the degrading performance of SFC dimension reduction in a high dimensional space. This method offers high efficiency but at the cost of low fidelity.

Our previous work [9] proposed PIRD DHT-based resource discovery mechanism in Internet-based distributed systems. PIRD builds a multi-dimensional space as in [8]. It relies on an existing locality sensitive hash technique in Euclidean spaces [14] to create a number of IDs for a resource, and then maps the resource to DHT nodes. In a system with a tremendous number of resource attributes, PIRD leads to high memory consumption and low efficiency of resource ID creation due to long resource vectors. Recognizing this drawback, we further developed optimized PIRD (OPIRD) by using LZW dynamic compression algorithm [15] to reduce the length of resource vectors.

III. Efficient, Flexible and High-fidelity Grid Information Service

A. Resource Attribute Transformation

To transform each attribute of a resource to an integer, we propose three attribute transformation algorithms with similarity preserving features. The first attribute transformation algorithm, denoted by *SHA*, applies a consistent hash function on each attribute of a resource to change the resource description to a resource vector. For example:
resource description A: Memory 512MB CPU 2GHz
resource vector A': 1945 6281 214 2015.
Because of the collision-resistant nature of consistent hash function, the same attributes will be transformed to the same integers. Thus, *SHA* preserves the similarity between resource descriptions to a certain extend.

The second attribute transformation algorithm, denoted by *Alphanumeric*, relies on an alphanumeric list to change a resource attribute to an integer. As shown in Figure 1, each resource attribute is represented by a 36-bit binary number. Each bit position of the list has the number of occurrences of its letter or digit in the attribute. For example, the resource vector of "Memory 512MB CPU 2GHz" generated is:
000010000000201001000000100000000000
010000000000100000000000000110010000
001000000000000100001000000000000000
000000110000000000000000010010000000.
This algorithm provides a certain degree of similarity preservation. However, if the difference of integers is used to evaluate the degree of similarity between resources, the letters and digits in higher significant bit positions have higher weights.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9
Memory	0	0	0	0	**1**	0	0	0	0	0	0	0	**2**	0	**1**	0	0	**1**	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0
512MB	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	**1**	0	0	0	0	0
CPU	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2Ghz	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	0	0	0	0	0	0

Fig. 1. The *Alphanumeric* attribute transformation.

The third algorithm combines the *Alphanumeric* algorithm with Hilbert space-filling curve (SFC) technique [13], denoted by *Hilbert*. The alphanumeric list can be regarded as a 36-dimension Cartesian space with each letter or digit representing a dimension. SFC maps points in the 36-dimensional Cartesian space into a domain of real numbers. *Hilbert* applies the Hilbert SFC hash function to the integer generated by the *Alphanumeric* algorithm. For example:
resource description A: Memory 512MB CPU 2GHz
resource vector A': 19 61 24 15.
Since the same resource attributes are transformed to the same integers, and more similar attributes are transformed to closer integers, *Hilbert* provides similarity preserving resource attribute transformation. In addition, *Hilbert* has higher

similarity preserving capability than *Alphanumeric* because it assigns the same weight to all bits in the intermediate 36-bit number when generating the final integers.

B. Locality-Sensitive Resource Translation

Using the introduced attribute transformation algorithms and min-wise independent permutations [11], we develop locality sensitive hash functions which do not need a pre-defined attribute list. We use SHA-LGS, Alpha-LGS and Hilbert-LGS to represent the new locality sensitive hash functions combing *SHA*, *Alpha* and *Hilbert* with min-wise independent permutations respectively.

Definition 1 A family of hash functions $\mathcal{F}$ is said to be a locality sensitive hash function family corresponding to similarity function $sim(A, B)$ if for any two sets A and B from the domain of hash functions, we have:

$$\mathbf{Pr}_{h\in\mathcal{F}}(h(A) = h(B)) = sim(A, B) \qquad (1),$$

where $\mathbf{Pr}$ is probability and $sim(A, B) \in [0, 1]$ is a similarity function [16, 17].

Definition 2 $\mathcal{F} \subseteq S_n$ is min-wise independent if for any set $X \subseteq [n]$ and $x \in X$, when permutation π is chosen at random in $\mathcal{F}$,

$$\mathbf{Pr}(min\{\pi(X)\} = \pi(x)) = \frac{1}{\mid X \mid} \qquad (2).$$

In other words, all the elements of any fixed set X have an equal chance to become the minimum element of the image of X under π [11].

Min-wise independent permutations provide an elegant construction of a locality sensitive hash function with the Jaccard set similarity measure:

$$sim(A, B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid} \qquad (3).$$

For example, the similarity between A and B, where
A=Memory | 512MB | CPU | 2GHz, and
B=Memory | 512MB | CPU | 3.5GHz, is $\frac{3}{5} = 0.6$.
By combining formulas (2) and (3), we get:

$$\mathbf{Pr}_{h\in\mathcal{F}}(min\{\pi(A)\} = min\{\pi(B)\}) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid} = sim(A, B).$$

Based on Definition 1, $min\{\pi(A)\}$ is locality sensitive hash function.

As the work in [11], LGS defines the min-wise independent permutations as:

$$\pi(x) = (ax + b) \bmod \hat{p},$$

where a and b are random integers, $0 < a \leq \hat{p}$ and $0 \leq b \leq \hat{p}$, and $\hat{p}$ is a prime number. Because the permutations can only be applied to values, we combine an attribute transformation algorithm and the min-wise independent permutations to build a locality hash function. Given a resource description A, using one of the attribute transformation algorithms introduced, LGS converts it into $A' = \{a_1, a_2, ..., a_l\}$, where $a_i (1 \leq i \leq l)$ is an integer and l is the number of attributes in the resource description. A locality sensitive hash function h_π is constructed as:

$$h_\pi(A') = min\{\pi(A')\} = min\{\pi(a_1), \pi(a_2), ..., \pi(a_l)\}.$$

That is, the hash function h_π applies the permutation π on each integer component in A' and then takes the minimum of the resulting elements. h_π is a locality sensitive hash function for resource vectors since

$$\mathbf{Pr}(h_\pi(A') = h_\pi(B')) = sim(A', B').$$

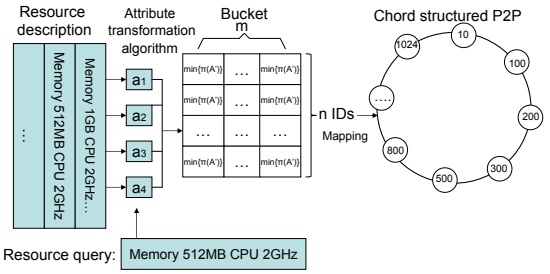


Fig. 2. Process of LGS operation.

Figure 2 illustrates the process of LGS operation. To build a family of hash function $\mathcal{F}$, LGS makes n groups of hash functions with each group having m number of h_π. Applying the $m \times n$ hash functions to a resource vector A', we get n buckets with each bucket having m hash values. A bucket represents a group of m hash values generated by $h_{\pi_i} (1 \leqslant i \leqslant m)$. LGS then executes XOR operation on the values in each bucket to get a final hashed value. Consequently, each record has n hash values, denoted by *IDs*.

$$ID_i = (min\{\pi_1(A')\} \wedge \ ...\, min\{\pi_m(A')\}) \bmod ID_{p2p},$$

where $1 \leq i \leq n$, and ID_{p2p} is the length of P2P key space. Similar groups of $h_{\pi_i}(A')$ $(1 \leqslant i \leqslant m)$ are transformed to close IDs by XORing their m hash values. Consequently, similar resources have close IDs. The IDs of a resource are its final keys for storing and searching the resource information in a DHT overlay.

C. DHT-based Grid Information Service

LGS is built on top of a DHT overlay to achieve multi-attribute resource searching. The overlay network provides two main functions: `Insert(key,object)` and `Lookup(key)` to store an object to a node responsible for the key, and to retrieve the object. We define that the resource predicate of a resource is represented in the form of $< v, ID, ip\_addr >$, where ip_addr is the IP address of the resource owner. Using the developed locality-sensitive hash function, a node produces n IDs of its resource "Memory 512MB CPU 2GHz Bandwidth 10Mbps",

$$ID_1, ID_2, ..., ID_n.$$

It then uses

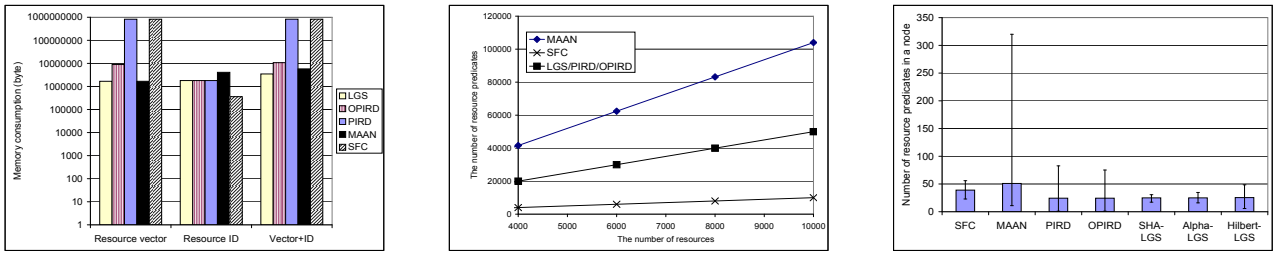
$$\texttt{Insert}(\texttt{ID}_\texttt{i}, < \texttt{v}, \texttt{ID}_\texttt{i}, \texttt{ip\_addr} >) \ (1 \leq i \leq n)$$

to insert the predicate of the resource to the P2P overlay.

In LGS, each grid node periodically inserts the predicates of its available resources to the system. Due to the similarity-preserving feature of the developed locality-sensitive hash functions, the predicates of similar resources will be stored in the same or close nodes. This facilitates similar resource searching in the grid information service. Note that for two resource vectors A' and B',

$$\mathbf{Pr}_{h\in\mathcal{H}}[h(A') = h(B')] = sim(A', B'),$$

this resource predicate storing method can save the predicates of similar resources to the same nodes with probability $\geq$



(a) Memory consumption (b) Num. of resource predicates (c) Predicate distribution balance

Fig. 3. Overhead of different grid information services.

$1-(1-p^m)^n$ [14].

A resource query is represented by a resource description. Assume the resource vector of a requester's query is A'. The requester first produces n IDs from A' using the locality-sensitive hash function. Note that the vectors of similar resources and the query could be hashed to the same IDs with high probability (i.e., $1-(1-p^m)^n$). Thus, by having these IDs as the DHT keys in the function

$$\texttt{Lookup}(\texttt{ID}_\texttt{i})\ (1 \leq i \leq n),$$

the resource requester will receive the predicates of its desired resources from the destination nodes. Since similar resources have close IDs, the nodes close to the destination nodes can be searched in order to avoid missing similar resources. Thus, a range value, R, is determined and the nodes whose IDs are in [ID-R,ID+R] are queried during resource searching.

IV. Performance Evaluation

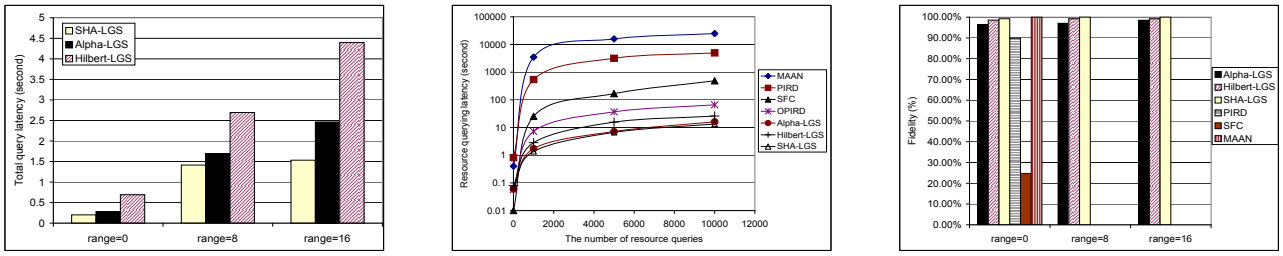
We designed and implemented a simulator for the evaluation of the LGS grid information service. We compared LGS with the direct mapping and one-point mapping systems, and PIRD/OPIRD [9] in terms of overhead, efficiency and fidelity. We used MAAN [4] to represent the direct mapping systems, and used SFC [8] to represent the one-point mapping systems, respectively. The number of nodes in Chord was set to 2048. The total number of attributes in the system was set to 20,591, and the total number of resources was set to 10,000. The number of resource queries was set to 100 unless otherwise specified. We set $m = 20$ and $n = 5$ in LGS. We include some of the experiment results in [9] for reference.

A. Overhead

Resource predicates stored in a node includes resource vectors and IDs. Figure 3(a) shows the total memory consumption in bytes for resource vectors and IDs. We can see that SFC and PIRD consume prohibitively more memory for resource vectors. Recall that SFC and PIRD regard each resource attribute as a dimension in a multi-dimensional space. Thus, the length of a resource vector is the number of total attributes in the system, regardless of the number of attributes the resource contains. Using LZW compression algorithm, OPIRD reduces the length of a resource vector but with an extra cost. LGS and MAAN generate a resource vector by transforming each attribute in a resource using a hash function. The length of a resource vector is the number of attributes in the resource. As a result, they consume much less memory for vectors. The figure also shows that for resource IDs, SFC leads to the least and MAAN leads to the most memory consumption. This is because that for a l-attribute resource, MAAN produces l IDs, LGS produces five IDs, while SFC produces one ID. Combining the memory for vectors and IDs, the total memory consumption of SFC and PIRD is much higher than OPIRD, whose memory consumption is higher than LGS and MAAN. This result confirms an advantage of LGS without building a multi-dimensional space.

Figure 3(b) shows the total number of resource predicates stored in the system versus the number of resources. We can see that LGS/PIRD/OPIRD generate fewer resource predicates than MAAN. Recall that LGS/PIRD/OPIRD change each resource to n hash values regardless of resource vector length. MAAN hashes each attribute of the resource, and stores the resource predicate in a node responsible for each hashed value. For a l-attribute resource, MAAN needs l messages for storing and serching l resource predicates. Therefore, MAAN generates many resource predicates for a resource with many attributes. SFC and LGS/PIRD/OPIRD weave all attributes of a resource to one and five ID(s) respectively. They need one and five message(s) for storing and searching a resource. Hence, SFC generates less resource predicates than LGS. The experiment results imply that LGS/PIRD/OPIRD need less node communication than MAAN and higher node communication than SFC for resource pooling and querying. However, the advantage of SFC is outweighed by its high memory consumption, and low fidelity of searching desired resources.

Figure 3(c) plots the average, the 1st and 99th percentiles of the number of resource predicates in a node. The average is the total number of resource predicates divided by the number of nodes having the resource predicates. Two observations can be made from the figure. First, the average number of resource predicates of MAAN is much higher than others, and that of SFC is higher than LGS. Due to the same reason observed in Figure 3(b), MAAN generates more resource predicates than others. It is surprising to see that SFC produces higher average number than LGS. Our experiment shows that SFC generates 258 different IDs totally, while LGS generates 2048 different IDs. Therefore, because of SFC's coarse-grained resource classification, only a fraction of the nodes in the system are responsible for resource predicates. In contrast,



(a) Query latency of LGS (b) Query latency of different services (c) Fidelity

Fig. 4. Efficiency and fidelity of different grid information services.

LGS spreads resource predicates among all nodes, leading to even load distribution.

Second, MAAN exhibits the largest variance, and LGS incurs the least variance. MAAN maps resource predicates to a DHT overlay based on each attribute, leading to much more resource predicates stored in some nodes while only a few stored in others. LGS regards all attributes of a resource as a single entity to generate n IDs, and distributes the resource predicates to the entire system based on the IDs. In addition, it incurs much less predicates. Therefore, its variance is not as significant as MAAN. SFC incurs higher variance than LGS, because it only relies on 258 nodes among 2048 nodes for all resource predicates. Though PIRD and OPIRD also regard a resource as an entity, they rely on a multi-dimensional space with a large number of dimensions to generate resource IDs. The high dimension may degrade the locality preserving feature of hashing, generating more the same or similar IDs. Thus, they exhibit larger variance than LGS. We also observe that Hilbert-LGS has slightly larger variance than SHA-LGS and Alpha-LGS. This is because Hilbert-LGS depends on SFC to produce resource IDs, leading to more the same or similar IDs. The results show that LGS can achieve more balanced distribution of load due to resource predicates maintenance and resource query response.

B. Efficiency

Recall that a range R can be used to query resources from nodes with ID $\in [ID - R, ID + R]$ in order to avoid missing similar resources. Figure 4(a) plots the resource query latency of SHA-LGS, Alpha-LGS and Hilbert-LGS when R equals to 0, 8 and 16 respectively. The latency does not include the routing latency in the network. We can see that the latency increases as the range increases. A larger range results in more located resources, which leads to longer time for filtering unsatisfying resources. We can also see that SHA-LGS leads to less latency than Alpha-LGS, which offers faster service than Hilbert-LGS. To change a resource description to a resource vector, SHA-LGS applies a consistent hash function to each attribute of a resource, Alpha-LGS maps the letter and digits in each attribute to an alphanumeric list, and Hilbert-LGS further transforms each identifier in the resource vector resulting from Alpha-LGS to a Hilbert number. The alphanumeric mapping takes longer time than the consistent hashing, hence Alpha-LGS leads to longer query latency. Since Hilbert-LGS has one more step of Hilbert hashing than Alpha-LGS, Hilbert-LGS needs longer latency than Alpha-LGS.

Figure 4(b) depicts the resource querying time of different services. MAAN leads to dramatically higher querying time than others. For a query consisting of l attributes, MAAN generates l queries for all attributes. For example, for a query "Memory 2048MB CPU 3GHz," MAAN sends four requests, collects the resource predicates for each attribute, and then merges the resource predicates to find satisfying resources. However, since there could be an enormous number of resources owning a single attribute, there will be a huge volume of resource predicates discovered in MAAN. Therefore, MAAN needs a very long time to prune unsatisfying resources in the merging phase. PIRD changes each resource to n IDs with locality-preserving feature, it does not need a long time for pruning unsatisfying resources, resulting in less latency than MAAN. By compressing the resource vector, OPIRD greatly reduces the latency of PIRD. SFC only produces one ID for each resource, leading to less latency than PIRD. However, the length of resource vectors in SFC is very long, which equals the total number of attributes in the system. Thus, SFC needs longer time for processing long resource vectors than OPIRD. Unlike these methods, LGS generates resource IDs without building a multi-dimensional space. Each resource has a vector with length equals to the number of attributes in the resource description. Consequently, LGS exhibits the fastest querying speed among all methods. The results show the benefits of avoiding building a multi-dimensional space to produce resource IDs.

C. Fidelity

A high-fidelity grid information service should locate most resources in the system satisfying a request. We use *satisfying resources* of a resource query to represent all resources in the system having more than 50% similarity with the query. We define *fidelity* metric as the total number of satisfying resources located divided by the total number of satisfying resources in the system. Figure 4(c) depicts the fidelity of each system. Since the range querying results of PIRD, SFC and MAAN are not comparable to the range querying results of LGS, we omit these results. Because PIRD exhibits higher fidelity than OPIRD, we also omit the results of OPIRD. From the figure, we can see that MAAN produces the highest fidelity, while SFC generates the lowest fidelity. Since MAAN finds all

resources that have any attribute in a request, and merges the discovered information for the requested resource, it will not miss any satisfying resource in the system. SFC's effectiveness of locality preserving is degraded in a high dimensional space, so it misses around 40% of satisfying resources. Recall that in LGS, similar resources are hashed to the same node with probability of $\geq 1-(1-p^m)^n$, so that LGS misses about 7% of satisfying resources. MAAN achieves high fidelity at the cost of dramatically high overhead and high querying latency. LGS achieves relatively high fidelity at a significantly low overhead and high efficiency. PIRD's fidelity is not as high as LGS and MAAN. This is because the locality sensitive hashing is applied to very long resource vectors, which affects the effectiveness of the locality preserving. As a result, some satisfying resources may be missed during querying. We can also observe that SHA-LGS's fidelity is slightly higher than Hilbert-LGS, which incurs slightly higher fidelity than Alpha-LGS. This is due to their different algorithms to transform resource attributes to integers. With the increase of searching range, more satisfying resources are located, and therefore each LGS method produces marginal higher fidelity.

V. Conclusions

By pooling together globally-scattered resources, grid computing makes petascale systems become more of a reality than a dream. A key hurdle to overcome in grid resource sharing is an efficient and effective grid information service. In spite of the efforts to develop the services, most of them lead to low efficiency while others are not effective in locating satisfying resources in an environment with a tremendous number of resource attributes. In addition, most services exhibit limited flexibility by relying on a pre-defined attribute list for resource description and offering only exact-matching services. This paper presents an efficient and high-fidelity grid information service, namely LGS. LGS constructs novel locality sensitive hash functions, and relies on the hash functions to cluster the data of resources with similar attributes to facilitate efficient resource searching. More importantly, it is effective in locating satisfying resources in an environment with an enormous number of resource attributes. Furthermore, it provides high flexibility by removing the requirement of a pre-defined attribute list for resource description and providing similar-matching service. LGS is built on a DHT overlay, which facilitates efficient resource data pooling and searching in grids. Experiment results demonstrate the efficiency and effectiveness of LGS in comparison with other services. It dramatically reduces overhead and yields significant improvements in efficiency, and provides high fidelity in locating similar resource. Its high efficiency, fidelity and flexibility are particularly attractive to the deployment of large-scale grid applications.

Acknowledgment

This research was supported in part by U.S. NSF grants CNS-0834592 and CNS-0832109.

References

[1] 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.

[2] B. Y. Zhao, L. Huang, and et al. Tapestry: An Infrastructure for Fault-tolerant wide-area location and routing. *J-SAC*, 12(1):41–53, 2004.

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

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

[5] M. Cai and K. Hwang. Distributed Aggregation Algorithms with Load-Balancing for Scalable Grid Resource Monitoring. In *Proc. of IPDPS*, 2007.

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

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

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

[9] H. Shen. PIRD: P2P-based Intelligent Resource Discovery in Internet-based Distributed Systems Corresponding. *JPDC*, 2008.

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

[11] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, (3):630–659, 2002.

[12] S. Suri, C. Töth, and Y. Zhou. Uncoordinated Load Balancing and Congestion Games in P2P Systems. In *Proc. of IPTPS*, 2004.

[13] 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.

[14] 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.

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

[16] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. of STOC*, pages 380–388, 2002.

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