LORM: Supporting Low-Overhead P2P-based Range-Query and Multi-Attribute Resource Management in Grids

Haiying (Helen) Shen, Amy Apon
Dept. of Computer Science and Engineering
University of Arkansas, Fayetteville, AR 72701
{hshen,aapon}@uark.edu

Cheng-Zhong Xu
Dept. of Electrical & Computer Engineering
Wayne State University, Detroit, MI 48202
czxu@wayne.edu

Abstract

Resource management is critical to the usability and accessiblity of grid computing systems. Conventional approaches to grid resource discovery are either centralized or hierarchical, and these prove to be inefficient as the size of the grid system increases. The Peer-to-Peer (P2P) paradigm has been applied to grid systems as a mechanism for providing scalable range-query and multi-attribute resource management. However, most current P2P-based resource management approaches support multi-attribute range queries at a high cost. They either depend on multiple P2P networks with each P2P network responsible for a single attribute, or they keep the resource information of all attributes in a single node. This paper presents a low-overhead range-query multi-attribute P2P-based resource management approach, LORM. Unlike other P2P-based approaches, it relies on a single P2P network and allocates resource information to different nodes based on resource attributes and values. Moreover, it has high capability to handle the large-scale and dynamic characteristics of resources in grids. Simulation results demonstrate the efficiency of LORM in comparison with other resource management approaches in terms of resource management overhead and resource discovery efficiency.

1 Introduction

Grid systems integrate computers, clusters, storage systems, instruments, and in general, can provide a highly available infrastructure for large scientific computing centers. Grids make possible the sharing of existing resources such as CPU time, storage, equipment, data, and software applications. Some grid computing systems are used for complex scientific applications that are time critical and comply with strict Quality of Service (QoS) rules. In many grid systems, resources are highly dynamic and vary significantly over time. Therefore, scalable and efficient resource management is critical to providing usability and accessibility for large-scale grid computing systems.

The resources required by applications are often described by a set of attributes (multi-attribute queries) such as available computing power and memory. A fundamental service of resource management is to locate resources across multiple administrative domains according to the attribute inputs. Traditional approaches to the task of resource location are to maintain either a centralized server or a set of hierarchically organized servers that index resource information. However, a centralized approach does not scale to a large number of grid nodes across autonomous organizations. Hierarchical approaches provide better scalability and failure tolerance by introducing a set of hierarchically organized servers and by partitioning resource information across different servers. However, they do not adapt well to the dynamic nature of grids, and support efficient communication between nodes for resource management.

Recently, the Peer-to-Peer (P2P) paradigm has been applied to grid systems for solving large-scale and dynamic resource management. P2P-based resource management approaches can be classified into two categories: 1) unstructured P2P approaches, and 2) Distributed Hash Table (DHT)-based approaches. The former depends on message flooding or random-walkers as the primary mechanism for searching for resources. Flooding mechanism results in a large volume of messages. Random-walkers reduce flooding by some extent, but it cannot guarantee successful resource locations. In contrast, the constrained length of the search path and the deterministic search results of DHTs make them very effective in large scale and dynamic resource management in grids. However, most DHT-based approaches support multi-attribute range queries by relying on multiple DHT networks with each DHT responsible for a single attribute. There is a high cost to maintaining a

number of DHT networks. Some other DHT-based approaches keep the resource information of all values for a specific attribute in a single node. This approach has high cost for resource searching due to the large size of the resource information directory. Moveover, it overloads nodes that maintain a huge amount of information and process resource queries.

To reduce the high overhead, we propose a DHT-based resource management approach with Low-Overhead, Range-query and Multi-attribute (LORM) features for grids. Unlike most current P2P-base resource management approaches, it is built on a single DHT called Cycloid [1]. Instead of collecting resource information of all values of an attribute in a single node, LORM lets each node be responsible for information of a specific attribute within a value range by taking advantage of the hierarchical structure of Cycloid. It has high efficiency by achieving balanced distribution of resource management overhead.

The remainder of this paper is structured as follows. Section 2 presents a concise review of representative resource management approaches for grid systems. Section 3 describes the DHT-based LORM resource management approach, focusing on its resource management framework and resource management algorithm. Section 4 analyzes the performance of LORM in both a static environment and a dynamic environment, including a comparison using a variety of metrics, and analyzes the factors affecting resource management performance. Section 5 concludes the paper and provides remarks on possible future work.

2 Related Work

There are many resource management approaches in grid systems. Some approaches are based on centralized or hierarchical client/server models. For example, Globus toolkit [2] uses an LDAP-based directory service named MDS [3] for resource registration and lookup. Systems such as Condor-G [4] and Nimrod/G [5] use the Globus toolkit to integrate with a grid computing environment for resource management. However, centralized approaches have inherent drawback of a single point of failure, and the centralized server(s) can become a registration bottleneck in a highly dynamic environment. To cope with these problems, a P2P middleware overlay can be used for resource management. These approaches can be classified into two categories: unstructured P2P-based approaches [6–10] and DHT-based approaches [11–16]. Iamnitch *et al.* [6] and Talia *et al.* [7] propose organizing the MDS directories in a flat, dynamic P2P network; Mastroianni *et al.* [8] and Puppin *et al.* [10] propose clusters with supernodes, Marzolla *et al.* [9] propose a tree structure, and Hu *et al.* [17] proposed an overlay based service discovery mechanism in grid. However, flooding, random walkers searching methods in unstructured P2Ps prevent these approaches from achieving high scalability. Moreover, the indeterministic search results in unstructured P2Ps cannot provide a guarantee of the desired resource result. As a successful model that achieves scalability, robustness, and deterministic data location, DHT has been widely adopted for resource discovery in grids.

Chen [18] and Spence [13] use a DHT overlay to efficiently manage dynamic grid computing resources. Recently, two important issues investigated in these systems are range queries and multi-attribute resource discovery. Range queries look for resources specified by a range of attribute values (e.g., a CPU with speed from 1:2GHz to 3:2GHz). Current approaches to achieve multi-attribute range query mainly can be classified into three groups: (1) those that adopt one DHT for each attribute, and process multi-attribute range queries in parallel in corresponding DHTs [12, 14, 16, 19] (2) those that pool together resource information of all values for a specific resource attribute in a single node [15]; (3) those that map the resource attribute and value in a resource information separately to one DHT, and process a query by searching them separately [11, 20]. The first group comes at a high maintenance overhead for multiple DHTs. The second group overloads directory nodes for maintaining resource information and processing resource query. Moreover, large directories lead to inefficiency in resource searching. The third group brings about inefficiency in resource information reporting and searching, and it produces more maintenance overhead for resource information by doubling resource information. In this paper, we will take advantage of the Cycloid [1]'s hierarchical structure to use a single DHT to realize multi-attribute range-query resource management with low overhead.

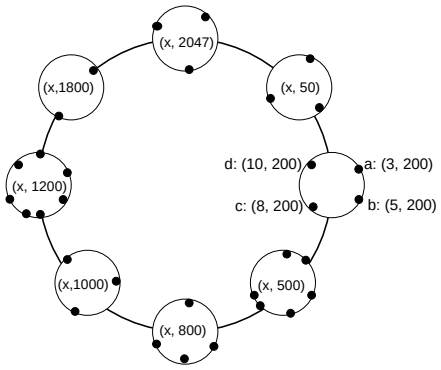


Figure 1. Cycloid partial routing links.

3 Range-Query and Multi-Resource Resource Management

3.1 Cycloid Overview

This section provides an overview of Cycloid DHT overlay networks followed by a high-level view of the LORM architecture. Cycloid is a lookup efficient constant-degree overlay. It achieves a time complexity of O(d) per lookup request by using O(1) neighbors per node, where $n{=}d \cdot 2^d$ nodes and d is dimension. Each Cycloid node is represented by a pair of indices $(k, a_{d-1}a_{d-2}\dots a_0)$, 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*. The largest cyclic index node in a cluster is called the *primary node* of the nodes at the cluster. All clusters are ordered by their cubical index mod 2^d on a large cycle. Figure 1 shows the partial routing links of a 11-dimensional Cycloid, where x indicates all possible cyclic index which ranges from 0 to 10. The Cycloid DHT assigns keys onto its ID space by the use of a consistent hashing function. For a given key, the cyclic index of its mapped node is set to its hash value modulated by d and the cubical index is set to the hash value divided by d. A key will be assigned to a node whose ID is closest to its ID. If the target node of a key's ID $(k, a_{d-1}\dots a_1a_0)$ is a participant, the key will be mapped to the node with same ID. If the target node is not a participant, the key is assigned to the node whose ID is first numerically closest to $a_{d-1}a_{d-2}\dots a_0$ and then numerically closest to k. The consistent hashing [21] produces a bound of $O(\log n)$ imbalance of keys between nodes, where n is the number of nodes in the system. Cycloid exhibits a more balanced distribution of key loads between the nodes. The balanced key load distribution helps LORM to prevent resource information imbalance, which is a severe problem in most grids centrally or hierarchically administered for resource management. Cycloid has APIs including `Insert(key, object)`, `Lookup(key)`. The APIs facilitate the collection of resource information and provide resource search functionality. For more information about Cycloid, please refer to [1]. Unlike most resource management approaches that depend on multiple DHTs for range-query and multi-attribute resource management, LORM relies on a single Cycloid with constant maintenance overhead. The goal of LORM is to address multi-attribute and range query resource discovery with low overhead and high efficiency.

3.2 Low-overhead Range-Query and Multi-Attribute Resource Management

This section introduces the framework of LORM based on Cycloid and related algorithms. The resource management problem is complicated by the heterogeneity of resources in terms of available CPU time, memory, available storage, network bandwidth, processing power, available software, and so on. Usually, the resources required by applications are described by specifying a set of attributes such as available computing power and memory. It is a challenge for a resource manager to effectively locate resources across widely dispersed domains based on a list of predefined attributes.

Currently, most approaches solve the multi-resource management problem using multiple DHTs with each DHT responsible for an attribute. To locate resources specified by several attributes, the resource manager uses multi-attribute queries and presents each query for a resource to the appropriate DHT and then concatenates the results in a database-like "join" operation. However, the construction and maintenance of multiple DHTs are costly, especially in dynamic environment. For example, suppose that there are q types of resource attributes. All together q DHTs are used. Although one node does not necessarily have all attributes, it is included in all DHTs. The number of figure tables that a node maintains is q. Each figure table, for overall n nodes, contains $\log n$ entries. Therefore, each node needs to maintain $q \times \log n$ neighbors. To address this problem, we propose LORM. In LORM, each node only needs to maintain seven neighbors in this example.

A computing resource has a specific value for each attribute, for example, "OS name=Linux", "CPU speed=1000MHz", and "Free memory=1024MB". Without loss of generality, we assume that each resource is described by a set of attributes with globally known types and values or ranges or string description. For example, Memory$\geq$2 or OpSys =="IRIX". We define *resource information* as information about available resources and resource queries. We use the function π_a to denote the value (range) or string description of a particular attribute a. Load information of a resource requester j is represented in a set of 3-tuple representation: $< a, \pi_a, ip\_addr(j) >$. The available resource information of node i is represented in the form of $< a, \delta\pi_a, ip\_addr(i) >$, in which $ip\_addr(i)$ denotes the IP address of node i, and $\delta\pi_a$ is the π_a of its available resource. Usually, the operation in resource management is to pool together information of available resources in a number of repository nodes, and direct resource requests to those nodes, which return the locations of desired resources to the requesters. The

repository nodes that get the information of available resource are called *directory nodes*, and the set of resource information stored in them is called a directory.

Resource	Hash
CPU	50
Memory	150
Disk	450
External memory	700
Software package	1000
Web service	1200
Bandwidth	1850
Database	2000

Figure 2. Resources hash values.

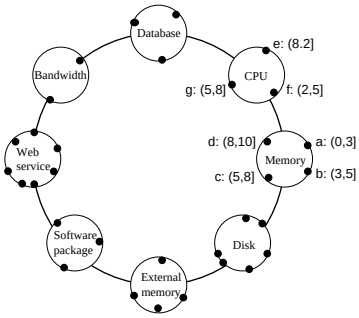


Figure 3. Resource information allocation in LORM.

A Cycloid consists of a number of clusters, which constitute a large cycle. LORM lets each cluster be responsible for the information of a type of attribute, and divides the information to nodes within the cluster based on resource value. The question is, how to achieve this objective efficiently? Recall that in a Cycloid ID, the cubical indices differentiate clusters, and the cyclic indices indicate different node positions in a cluster. Based on the Cycloid topology, LORM uses cubical indices to represent different resource attributes a such as "Memory size" and "OS name", and uses cyclic indices to represent different resource values π_a such as "2" or string description of resource such as "IRIX". Specially, LORM assigns each piece of resource information a Cycloid ID ($\pi_a\%d$, H_a). Its cubical index H_a is the consistent hash function value of the resource attribute. Its cyclic index $\pi_a\%d$ is the resource value or the consistence hash value of string description mod d. A node reports its available resources to the system periodically using the Cycloid interface `Insert(rescID, rescInfo)`. Based on the key assignment policy in Cycloid, in which a key is assigned to a node with closest ID to key's ID, the information of the same attribute will be mapped to the same cluster. Within each cluster, each node is responsible for the information of a resource whose cyclic index falls into the key space sector it supervises. For example, for the resources and their hashed values listed in Table 2, the resource information will be stored in the Cycloid in Figure 1 as in Figure 3. In the clusters responsible for memory and CPU, each node is responsible for the resource information in its range as illustrated. For instance, the resource information of node i $<mem, 2G, ip\_addr(i)>$ has resource ID ($2\%11$,H_{mem})=(2,200), and it will be routed to and stored in node a. To simplify the description, we use the following notation: let A denote the set of attributes in the overall schema of the application. A_Q denotes the set of attributes in a query Q. A node uses `Lookup(rescID)` to query for resources, and the query is routed to the directory node for the desired resource. A multi-attribute query Q is composed of a set of sub-queries on single attributes A_Q, and the sub-queries are processed in parallel. For example, when a node k needs a multiple-attribute resource, say 1.8GHz CPU and 2GB memory, it sends requests `Lookup(`$1.8\%d$`,`H_{cpu}`,`$<cpu, \pi_{cpu}, ip\_addr(k)>$`)` and `Lookup(`$2\%d$`,`$U_{mem}$`,`$<mem, \pi_{mem}, ip\_addr(k)>$`)`, which will be resolved in parallel. The queries will arrive at node a and node e, which reply to the requester node k with the requested resource information $< mem, \delta\pi_{mem}, ip\_addr(i) >$ where $\delta\pi_{mem} = 2$, and $< cpu, \delta\pi_{cpu}, ip\_addr(j) >$ where $\delta\pi_{cpu} = 1.8$. The requester node then concatenates the results in a database-like "join" operation based on ip_addr. The results are the ip_addr of nodes which have desired resource by the requester. For range queries such as cpu$\geq$1.8GHz and memory$\geq$2GB, in addition to responding with satisfied resource information in their own directories, node a and e forward the resource queries to their immediate successors in their own clusters. The successors check their own directories, response satisfied resource information to the requester, and forward the queries to their immediate successors in their own clusters. This process is repeated until a successor has no satisfied resource information. If the requested resource range is less than a value, then nodes forward queries to their predecessors. If the queries have lower and upper bounds such as 1GHZ$\leq$cpu$\leq$1.8GHz and 1GB$\leq$memory$\leq$2GB, the queries will be forwarded in both directions. Compared to other DHT-based approaches, this approach reduces the searching scope from n to d. Algorithm 1 shows the pseudocode in LORM resource management. Cycloid has a self-organization mechanism to maintain its structure, which helps LORM to handle dynamism. With this mechanism, instead of relying on specific nodes for resource information, resource information is always stored in a node responsible for the ID region where the information ID locates, even in dynamic situation, and the `Lookup(rescID)` requests will always be forwarded to the node that has the required resource information.

Algorithm 1 Pseudo-code for the operation of node i in LORM resource management.

```
1: //periodically report resource information of its available
   resources with attributes A={a1, a2 ... am}
2: for each a in {a1, a2 ... am} do
3:    get consistent hash values of attribute a: H_a,
4:    get the value or the consistent hash value of string de-
      scription: δπ_a%d
5:    get the rescID=(δ π_a%d, H_a)
6:    use DHT function:
      Insert(rescID,<a, δπ_a, ip_addr(i)>)
7: end for
8:
9: //to request resources using a multi-attribute range-queriy,
   Q={(a1, π_a1), (a2, π_a2) ... (am, π_a)}
10: for each a in {(a1, π_a1), (a2, π_a2) ... (am, π_a)} do
11:    get consistent hash values of attribute a: H_a,
12:    get the value or the consistent hash value of string de-
       scription: π_a%d
13:    get the rescID=(π_a%d, H_a)
14:    use DHT function:
       lookup(rescID,<a, π_a, ip_addr(i)>)
15: end for
16:
17: //after receive response of resource requests
18: concatenates ip_addr, get the results ip_addr(j),
    ip_addr(k) ...
19: ask resources from node node j, k ... in parallel
```

4 Performance Evaluation

This section presents the performance evaluation of LORM. As we mentioned in Section 2, there are mainly three approaches for grids resource management depending on DHTs. We use Mercury [14], SWORD [15], MAAN [11] to represent each of the approaches, and compare LORM with them. Mercury maps resource value to each DHT with each DHT responsible for an attribute. SWORD maps attribute in a flat DHT, MAAN maps attribute and value separately to a flat DHT, and LORM maps attribute and value to two levels in a hierarchical Cycloid. To be comparable, we use Chord for attribute hub in Mercury, and we replace Bamboo DHT with Chord in Sword.

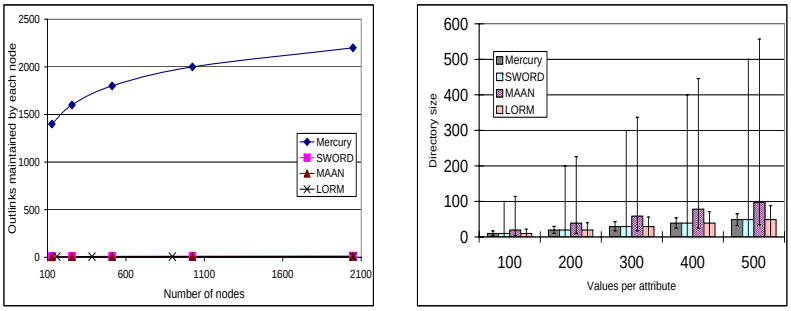
We have designed and implemented a simulator in Java for the evaluation of LORM, Mercury, Sword and MAAN. The dimension is 8 in the Cycloid, and 11 in Chord, and each DHT has 2048 nodes. We assume there are 200 resource attributes, and each attribute has 500 values. We used Bounded Pareto distribution function to generate resource values owned by a node and requested by a node. In the experiment, when a node needs to request resources, it chooses resource attributes randomly.

4.1 Maintenance Overhead

In DHT overlays, each node needs to maintain a number of neighbors in its routing table. Therefore, the routing table size or the number of outlinks a node maintains constitutes a large part of the DHT overlay maintenance overhead. Figure 4(a) plots the outlinks maintained by each node in different resource management approaches. It shows that Mercury has dramatically more outlinks per node than others. Recall that Mercury has multiple DHTs with each DHT responsible for each resource attribute, so that each node has a routing table for each DHT, and it has a total number of outlinks equal to the routing table size times the DHT number. It means that in Mercury each node needs much higher overhead to maintain its outlink than others. The experiment results further show that the outlinks of SWORD and MAAN increase with network size, while LORM keeps the result fixed at 7. It is because SWORD and MAAN are built on Chord whose routing table size is $\log n$. Their outlinks per node grow exponential as n increases. LORM is built on Cycloid, which is a constant-degree DHT regardless of the network size. The results illustrate that LORM has significantly less maintenance overhead, and much higher scalability compared with Mercury.

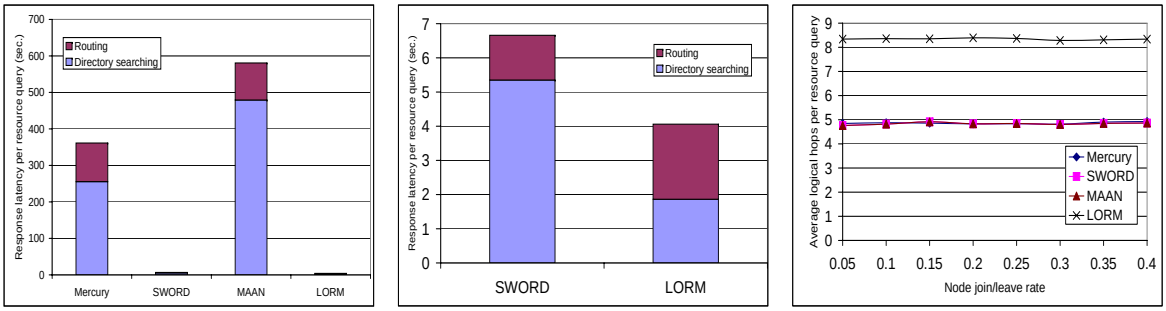
In addition to the outlinks, a directory node also needs to maintain resource information. It is desirable to distribute the information among nodes evenly so that the information maintenance overhead as well as resource management load can be distributed among nodes to avoid bottlenecks. The directory size is a metric of the resource information distribution. Figure 4(b) plots the average and the 1th and 99th percentiles of directory sizes. Observations can be made from the figure. First, the average size of MAAN is twice of others. Recall that MAAN separate resource attribute and value, and stores their information separately. As a result, its total resource information is doubled. Second, MAAN and SWORD exhibit significantly larger variance than Mercury and LORM. MAAN and SWORD classify resource information to directory nodes based on resource attribute. As there are 200 resource attributes, the information are accumulated in 200 nodes among 2048 nodes, leading to a very large directory size in some nodes while 0 size in others.

On the other hand, Mercury use a DHT for each attribute, and classifies resource information based on value in each DHT. The widespread value ranges help to distribute resource information evenly. LORM is based on Cycloid. It lets different clusters responsible for resource information based on resource attribute and further allocates information to a node based on its



(a) Outlinks per node with Mercury (b) Directory size

Figure 4. Overhead in different resource management approaches.



(a) Processing time (b) Processing time of SWORD & LORM (c) Performance with dynamism

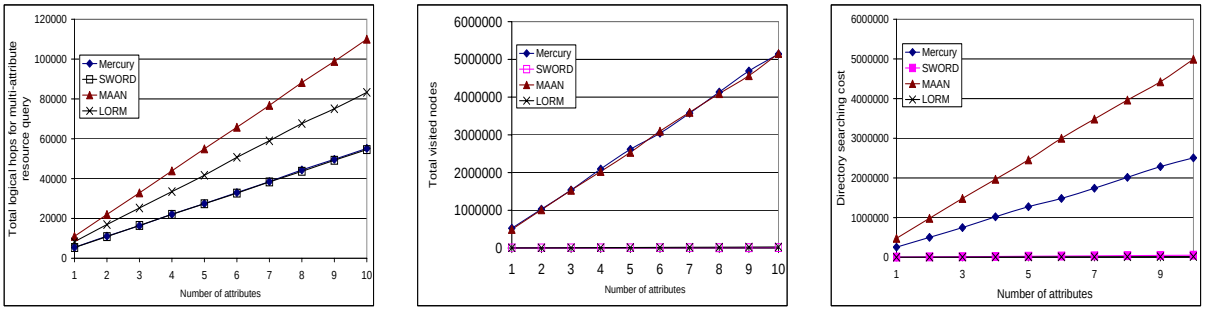
Figure 5. Efficiency of different resource management approaches.

range. Cycloid's more balanced key load distribution helps LORM achieve balanced information distribution. Therefore, Mercury and LORM can achieve more balanced distribution of load due to resource information maintenance and resource management operation. Since that each node in Mercury has dramatically more outlinks to maintain, LORM is the best with regards to overhead and load distribution.

4.2 Efficiency of Resource Management

An experiment was designed to evaluate the efficiency of different resource management approaches. We randomly chose 100 nodes and let each node send 10 resource queries. We varied the attribute numbers from 1 to 10 with 1 increase in each step. Figure 6(a) plots the total logical hops for resource queries with the number of attributes. We can observe that MAAN has higher results than LORM, and LORM has higher results than SWORD and Mercury. The higher hop count of Cycloid is due to its time complexity of lookups. Chord has a time complexity of $O(\log n)$ per query, and Cycloid has a time complexity of $O(d)$ per query. Because MAAN has two queries for each requested resource, resource attribute and range, it doubles the logical hops for each resource query.

For range resource query, after the destination node is reached, other nodes need to be probed for the resources in the specified range. We define the sum of destination nodes and probed nodes as visited nodes. This metric represents resource location efficiency. The more visited nodes, the less efficiency. Figure 6(b) plots the number of visited nodes versus the number of attributes. We can observe that Mercury and MAAN have tremendously more visited nodes than SWORD and LORM. Recall that Mercury and MAAN accumulate resource information based on attribute value, which spreads along the entire DHT ID space. They may need to probe nodes along a very long ID space. On the other hand, SWORD accumulates resource information based on node attribute name. All information of a specified attribute name is in the destination, and no nodes need to be probed. LORM stores resource information of a specified attribute name in a cluster, and only nodes in the cluster should be probed. As a result, SWORD and LORM have much less cost for range query than Mercury and MAAN. However, SWORD achieves it at a cost of high information maintenance overhead in directory nodes, and high directory searching cost. To verify this, we record the directory size of each visited node, and show the sum of the sizes in Figure 6(c) to show the directory searching cost. We can observe that SWORD



(a) Logical hops per query (b) Visited nodes (c) Directory searching cost

Figure 6. Searching cost in different resource management approaches.

has higher searching cost than LORM due to its large directory. Although LORM needs to probe nodes in a cluster, because of balanced information distribution, it still has less directory searching cost than SWORD. We can also observe that MAAN and Mercury have significantly higher cost, and MAAN has higher cost than Mercury. It is because a large number of nodes need to be probed in MAAN and Mercury. The doubled directory size in MAAN leads to its higher searching cost. Considering that the number of visited nodes is significantly larger than the routing hops, we can conclude that LORM is most efficient in terms of the total searching cost for resource queries.

Cost aside, speed of resource query response latency is another important metric to measure the efficiency of a resource management approach. We assume that each node communication takes 0.2 second, and each directory searching takes 0.01 second times the directory size. Figure 5(a) and (b) show response latency for all resource queries. We can observe that Mercury and MAAN take significantly longer response latency than SWORD and LORM, and the latency of MAAN is longer than Mercury. It is shown that Mercury and MAAN take the same time for routing, and MAAN has longer directory searching latency than Mercury. They have the same routing latency is because all queries for all attributes are processed in parallel. However, MAAN has larger directory size due to the doubled resource information, leading to longer latency for directory searching. From Figure 5(b), we can see that LORM has a shorter response latency than SWORD, though it has longer latency for routing. It is because Cycloid has a little longer query routing path length than Chord, and LORM needs to probe nodes in a cluster for range queries while SWORD does not need to probe nodes. However, LORM has more balanced resource information distribution, while SWORD accumulate information of a specified attribute to a single node, so LORM has less directory searching latency than SWORD. The results show that LORM is most efficient in resource query processing and response.

4.3 Dynamism-Resilient Resource Management

The results show that LORM achieves high performance along with cost reduction and efficiency in comparison to Mercury, SWORD and MAAN in a static situation. This section evaluates the efficiency of the LORM in dynamic environment. In this experiment, there are 10000 resource requests. The item join/departure rate is modeled by a Poisson process with a rate R of 0.4. That is, there is one item join and one item departure every 2.5 seconds. Node interarrival rates range from 0.1 to 0.5, with 0.1 increment in each step. Results show that there are no failures in all test cases. Figure 5(c) shows the average number of logical hops of lookup operations in different resource management approaches as the node join/leave rate R changes. We can see that the measured number of hops in dynamism is very close to the results in Section 4.2 and does not change with the rate R. The results show that with the help of the DHT maintenance mechanism, LORM can solve resource queries in a dynamic environment.

5 Conclusions

Grids employ distributed computational and storage resources to solve large-scale problems in science, engineering, and commerce. Resource management is a critical issue for grid systems in which applications are composed of hardware and software resource. This paper presents a DHT-based Low-overhead Range-query Multi-resource manager, LORM, which is built on Cycloid DHT. Unlike most previous resource managers which depend on multiple DHTs with each DHT responsible for a resource, LORM relies on a single DHT with constant maintenance overhead to achieve multi-

resource management with low overhead. It avoids bottlenecks by achieving a balanced distribution of load due to resource information maintenance as well as resource management operation itself. Furthermore, Cycloid's structure helps LORM to deal with dynamic node changes and variation of resource availability. Simulation results show the superiority of LORM in comparison with a number of other representative resource management approaches in terms of overhead cost and efficiency of range-query and multi-attribute resource management. In grids, resources are geographically distributed and owned by different organizations. Hence, we plan to further explore and elaborate upon the LORM design to discover resources based on node locality.

Acknowledgments

This research was supported in part by U.S. Acxiom Corporation and NSF grant MRI-0421099.

References

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

[2] I. Foster and C. Kesselman. Globus: a metacomputing infrastructure toolkit. *HPCA*, 2:115–128, 1997.

[3] I. Foster C. Kesselman G. Laszewski W. Smith Fitzgerald, S. and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. of HPDC*, pages 365–375, 1997.

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

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

[6] A. Iamnitchi and I.T. Foster. A peer-to-peer approach to resource location in grid environments. In J. Schopf J. Weglarz, J. Nabrzyski and M. Stroinski, editors, *Grid Resource Management*. Kluwer, 2003.

[7] D. Talia and P. Trunfio. In L. Grandinetti, editor, *Grid Computing: The New Frontier of High Performance Computing, Advances in Parallel Computing*. Elsevier Science, 2005.

[8] C. Mastroianni, D. Talia, and O. Verta. A superpeer model for building resource discovery services in grids: Design and simulation analysis. In *Proc. of EGC*, pages 132–143, 2005.

[9] M. Marzolla, M. Mordacchini, and S. Orlando. Resource discovery in a dynamic grid environment. In *Proc. of DEXA Workshop*, pages 356–360, 2005.

[10] D. Puppin, S. Moncelli, R. Baraglia, N. Tonelotto, and F. Silvestri. A grid information service based on peer-to-peer. In *Proc. of Euro-Per*, pages 454–464, 2005.

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

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

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

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

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

[16] S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Range queries over DHTs. Technical Report IRB-TR-03-009, Intel Corporation, 2003.

[17] C. Hu, Y. Zhu, J. Huai, Y. Liu, and L. Ni. S-Club: An Overlay Based Efficient Service Discovery Mechanism in CROWN Grid. *KAIS*, 2006.

[18] S. Chen, X. Du, F. Ma, and J. Shen. A grid resource management approach based on P2P technology. In *Proc. of HPC Asia*, 2005.

[19] D. Talia, P. Trunfio, J. Zeng, and M. Högqvist. A DHT-based Peer-to-Peer framework for resource discovery in grids. Technical Report TR-0048, Institute on System Architecture, CoreGRID - Network of Excellence, 2006.

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

[21] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and Panigrahy R. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proc. of STOC*, pages 654–663, 1997.