Locality-Preserving Clustering and Discovery of Resources in Wide-Area Distributed Computational Grids

Haiying Shen*, *Member, IEEE,* Kai Hwang, *Fellow, IEEE*

Abstract

In large-scale computational Grids, discovery of heterogeneous resources as a working group is crucial to achieving scalable performance. This paper presents a resource management scheme including a hierarchical cycloid overlay architecture, resource clustering and discovery algorithms for wide-area distributed Grid systems. We establish program/data locality by clustering resources based on their physical proximity and functional matching with user applications. We further develop dynamism-resilient resource management algorithm, cluster-token forwarding algorithm and deadline-driven resource management algorithms. The advantage of the proposed scheme lies in low overhead, fast and dynamism-resilient multi-resource discovery. The paper presents the scheme, new performance metrics, and experimental simulation results. This scheme compares favorably with other resource discovery methods in static and dynamic Grid applications. In particular, it supports efficient resource clustering, reduces communications cost, and enhances resource discovery success rate in promoting large-scale distributed supercomputing applications.

Keywords: Grid computing, resource discovery, DHT overlays, program/data locality, clustering techniques, and scalability.

1 Introduction

The popularity of the Internet as well as the availability of powerful computers and high-speed network technologies have led to what is popularly known as Grid computing. Grid computing leverages a high-degree of resource sharing in a large-scale distributed network environment. It enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources including supercomputers, storage systems, data sources, and specialized devices, and thus benefits a variety of applications such as collaborative engineering, data exploration, high-throughput computing, and distributed supercomputing.

Heterogeneous Grid resources owned by different organizations spread throughout the Internet or wide-area networks. To discover useful resources for a given application, most existing Grids apply centralized or hierarchical control [1–3]. However, these Grid resource management packages have limitations in supporting large-scale dynamic Grid applications with a varying demand of resources.

Overlay networks based on distributed hash tables (DHTs) have been suggested to manage large-scale Grid resources [4]. DHT overlay networks [5–8] map files to the nodes in a network based on a consistent hashing function [9]. Most of the DHT overlays require $O(\log n)$ hops per lookup request with $O(\log n)$ neighbors per node, where n is the network size. A computing resource is always described by a *resource type (i.e. functionality)* such as CPU and memory, and *resource attribute* indicating the quantity and special application requirement. To use a DHT overlay for resource discovery in a Grid system, all Grid nodes are organized into a DHT overlay. The descriptors of available resources are regarded as files and are distributed among the nodes. Resource queries are regarded as file lookups and are routed to the nodes having the descriptors of the required resources. Therefore, DHT overlays map the resource providers and consumers in Grids in a distributed manner.

In a wide-area Grid system, resource sharing and communication among physically close nodes enhance application efficiency. In addition, the problem of increasing complexity in using heterogeneous resources needs to be solved. Different resources, such as CPU and memory, are always jointly requested and used. Resource clustering based on functional matching with the demands of user application facilitates a user's resource discovery. We use *program/data locality* to represent the phenomenon in which resources are proactively clustered so that a node can always locate resources that (1) are physically close to itself, and (2) satisfy required multiple functionalities (e.g., CPU and Memory) in its neighborhood on the overlay. Therefore, after a node locates the CPU resource, it can easily find the memory resource in the nodes nearby, so that it does not need to search each of its required resources individually in the system-wide scope. It is desirable to develop a resource management scheme that is able to preserve the program/data locality.

However, the adoption of DHT overlays in most current resource management schemes [10–19] cannot preserve program/data locality. First, direct DHT construction on a Grid system breaks the physical proximity relationship of nodes in the underlying IP-level topology. That is, two nodes which are close in the DHT overlay are not necessarily close nodes in the underlying IP-level topology. Second, a node cannot locate its required resources with multiple functionalities in its neighborhood on the overlay. In current schemes, if a node needs m resources, it must send out m lookup messages, each

* *Corresponding Author. Email: shenh@clemson.edu.*

- *H. Shen is with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. K. Hwang is with the Department of Electrical Engineering and Computer Science, University of Southern California, Los Angeles, CA 90089.*

of which traverses $O(\log n)$ hops in the system-wide scope.

We desire to have a Grid resource management scheme that is program/data locality-preserving (locality-preserving in short), and also highly scalable and dynamism-resilient. To meet this demand, this paper extends from previous work [8, 20, 17] on a locality-preserving resource management scheme in Grid systems. The proposed scheme includes a new hierarchical cycloid overlay (HCO) architecture, resource clustering and discovery algorithms for efficient and scalable resource discovery. We call this scheme HCO. It establishes program/data locality by clustering resources based on their physical proximity and functional matching with user applications. Thus, HCO enables a node to not only jointly and easily find its required multiple resources in nearby nodes, but also find resources physically close to itself. We further develop dynamism-resilient resource management algorithm, cluster-token forwarding algorithm and deadline-driven resource management algorithms for HCO. The advantage of the HCO scheme lies in its low-overhead, fast and dynamism-resilient multi-resource discovery. The main contribution of this work is summarized in blow:

- Program/data locality-preserving resource clustering and discovery. HCO collects physically close resources into a cluster and connects the clusters based on their physical distances. It further groups the resources within a cluster based on resource functionalities. Thus, a node can always find physically close resources with required functionalities in its cluster or nearby clusters, leading to low-overhead and fast resource discovery.
- Cluster-token forwarding algorithm. The algorithm takes advantage of HCO's property of program/data locality-preserving to combine request messages together in resource clustering and discovery without compromising the resource discovery efficiency, leading to low overhead resource discovery.
- Deadline-driven resource management algorithm. The algorithm considers different priorities of resource requests in resource allocation and message forwarding, leading to on-time response for resource discovery.
- Comprehensive simulations validate the analytical results and confirm the advanced performance of the HCO scheme in comparison with previous resource discovery methods, and show the effectiveness of the cluster-token forwarding algorithm and deadline-driven resource management algorithm.

Within the authors' knowledge, the HCO scheme is the first work that preserves the program/data locality in Grid resource management for high scalability and efficiency. The rest of this paper is structured as follows. Section 2 presents a concise review of representative resource management approaches for Grids. Section 3 specifies the HCO architecture and applications. Section 4 presents the locality-preserving properties of HCO, the resource discovery and clustering algorithms, the algorithm to deal with dynamism, and the randomized probing and cluster-token forwarding algorithms. Section 5 reports the simulation experimental results in both static and dynamic network environments. The final section concludes with a summary of contributions and a discussion on further research work.

2 Related Work

To support multi-resource queries, Mercury [10] uses multiple DHT overlays. It uses one DHT for each resource, and processes multi-resource queries in parallel in corresponding DHT overlays. However, depending on multiple DHT overlays leads to high overhead for DHT maintenance. One group of approaches [11, 12, 14, 18, 19] organize all Grid resources into one DHT overlay, and assign all descriptors of one resource to one node. Multi-Attribute Addressable Network (MAAN) [11] extends Chord to support multi-resource queries. MAAN maps attribute values to the Chord identifier space via uniform locality preserving hashing. It uses an iterative or single attribute dominated query routing algorithm to resolve multi-resource based queries. To facilitate efficient range queries, Andrzejak and Xu proposed a CAN-based approach for grid information services [14]. SWORD [12] relies on one DHT to locate a set of machines matching user-specified constraints on both static and dynamic node characteristics. Cai and Hwang [18] proposed a Grid monitoring architecture that builds distributed aggregation trees (DAT) on a structured P2P network. Most of the single DHT-based approaches assign one node to be responsible for all descriptors of one resource, leading to an imbalanced distribution of workload. Also, if one of the nodes fails in dynamism, many descriptors will be lost at a time.

One DHT with n nodes needs a certain overhead, say o, for its structure maintenance. m DHTs formed by the n nodes lead to mo total overhead for structure maintenance. By using a single DHT, HCO generates much lower maintenance overhead than the methods based on multiple DHTs. HCO is more dynamism-resilient than the methods based on a single DHT by distributing resource descriptors of one resource among a number of nodes. More importantly, few of the current approaches can achieve the program/data locality to facilitate low-overhead and quick resource discovery. HCO is novel in that it establishes program/data locality, which enables users to discover their required physically close resources from their nearby nodes on the DHT overlay. The design of HCO facilitates the development of cluster-token forwarding algorithms to further improve efficiency. These features contribute to the high scalability and efficiency characteristics of HCO in Grid resource management.

The resource management and information services in P2P networks share similarity with resource management services in Grids in terms of data collection and search. One group relies on tree structure. SOMO [16] performs resource management in DHT networks by embedding a tree structure into a DHT structure. Ferry [21] is an architecture for content-based publish/subscribe services. It also exploits embedded trees in the underlying DHT to collect and classify data for subsequent data discovery service. In the tree structure, information is gathered from the bottom and propagates towards the root, and disseminated by trickling downwards. However, tree structure is not resilient to churn.

Some approaches [13, 17] focus on weaving all attributes of a resource into one or a certain number of IDs for resource clustering and discovery in a DHT overlay. Squid [22] uses a dimensionality reducing indexing scheme based on space filling curves to map the multi-dimensional information space to physical peers while preserving lexical locality. However, dimension reduction may generate false positives in information searching.

Hierarchical Bloom filter Arrays (HBA) [23] is an efficient and distributed scheme that maps filenames to metadata servers for file mapping and lookup. In the advertisement-based P2P search algorithm (ASAP) [24], nodes advertise their contents. After receiving a request, a node locates the destination nodes by looking up its local advertisement repository. In the Differentiated Search algorithm [25], nodes with high

query answering capabilities have higher priority to be queried, which leads to the reduction of search traffic by reducing queried nodes. Zhang and Hu [26] proposed partial indexing algorithm to assist data search. Li *et al.* proposed semantic small world (SSW) [27] to facilitate highly efficient semantic-based searches in P2P systems. The PASH [28] protocol can choose proper search methods to reduce query traffic cost and response time by dynamically estimating the content popularity. In order to enhance the successful content retrieval rate and decrease the costs, Pagani *et al.* [29] proposed a distributed infrastructure, in which peers are organized to mirror the semantic relations among contents.

3 Hierarchical Cycloid Overlay Network

HCO is built by extending the cycloid overlay [8]. We first describe cycloid followed by a description of the HCO architecture. Cycloid is a lookup efficient overlay network generalized from the *cube-connected cycles* (CCC) [30]. A d-dimensional cycloid is built with at most $n{=}d \cdot 2^d$ nodes. Like CCC, a cycloid has a constant node degree equals to its dimension d. The upper part of Figure 1 shows a 11-dimensional cycloid. In general, it takes at most $O(d)$ steps to lookup a file. The cycloid ID of a node or an object is represented by a pair of indices (id_s, id_l) ($id_s{\in}[0, d-1]$ and $id_l{\in}[0, 2^d-1]$) ($_s$ and $_l$ means small and large cycle respectively), where id_l is a cubical index representing the cluster that a node or an object locates and id_s is a cyclic index representing its position within a cluster. The nodes with the same id_l are ordered by their id_s on a small cycle called a *cluster*. All clusters are ordered by their id_l on a large cycle. The node with the largest id_s in a cluster is called the *primary node* of the cluster.

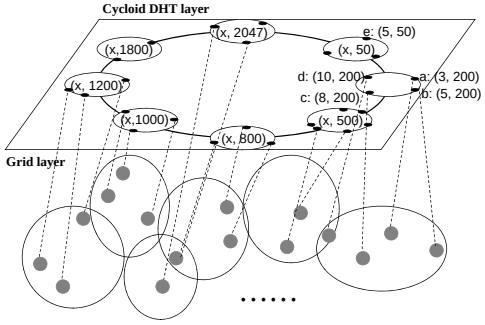


Fig. 1. A HCO architecture built on Grid resource clusters. A 11-dimensional cycloid overlay is shown to manage at most 2048 clusters (only 6 are shown). Nodes at different cycles in the cycloid manage different resource clusters in order to preserve program/data locality in distributed Grid computing (cluster mapping shown by dashed lines).

The distance between two IDs (d_{s_1}, d_{l_1}) and (d_{s_2}, d_{l_2}), denoted by $|ID_1{-}ID_2|$, is calculated by

$$((id_{s_1} - id_{s_2})\%d, (id_{l_1} - id_{l_2})\%2^d) = (\Delta id_s, \Delta id_l). \quad (1)$$

The distance is firstly measured by Δid_l (i.e., distance between clusters) and secondly Δid_s (i.e., distance between nodes in one cluster). The *logical distance* between node i and node j in an overlay equals $|ID_i - ID_j|$. For example, in Figure 1, the distances from node a=(3,200) to b=(5,200), d=(8,200) and e=(5,50) are (2,0), (5,0) and (2,150), respectively. Therefore, a=(3,200) is closer to b=(5,200) and d=(8,200) than e=(5,50) judging by Δid_l. From the figure, we can see node b and d are in the same cluster as a, while node e is in a different cluster. Also, a=(3,200) is closer to b=(5,200) than d=(8,200) judging by Δid_s. From the figure, we can see that b is closer to a than d in the same cluster. Cycloid assigns an object to the node whose ID is closest to its ID. It provides two main functions: `Insert(ID,object)` stores an object to a node responsible for the ID and `Lookup(ID)` retrieves the object through DHT-based searching. Each node maintains a routing table recording its neighbors in the overlay network for object lookups. Like all other DHTs, when a node joins in cycloid, it generates its cycloid ID and relies on a bootstrap node to find its neighbors and connect to them. Please refer to [8] for more details of cycloid.

TABLE 1
Notations.

Notation	Meaning
(id_s, id_l)	the ID of a node or object. id_s: cyclic index, id_l:cubical index
H_i	consistent hash value of node i's IP address
$\mathcal{H}_i$	Hilbert number of node i indicating node proximity
$ID_i{=}(H_i, \mathcal{H}_i)$	node i's ID
H_r	consistent hash value of resource r
$ID_r{=}(H_r, \mathcal{H}_i)$	resource ID for resource r in node i
D_r	resource descriptor for resource reports and resource requests

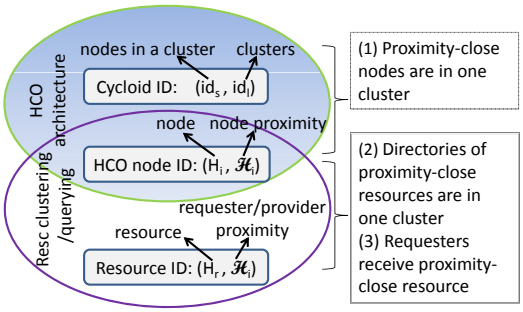


Fig. 2. IDs in HCO architecture and resource clustering and querying.

We present below the architecture and processing layers of HCO, which is a DHT-based hierarchy for locality-preserving Grid resource clustering and discovery. Figure 1 shows an example of the HCO architecture. In the figure, the resource nodes are shown at the bottom Grid layer. Various nodes are grouped into different clusters based on their physical proximity. All resource discovery operations are conducted in the overlay layer in a distributed manner. One major challenge in resource clustering is to keep the logical proximity and physical proximity of resource nodes consistent. A landmark clustering is adopted to generate proximity information [31, 32]. We assume that m landmark nodes are randomly scattered in the network. Each node measures its physical distances to the landmark nodes. A vector of distances $< d_1, d_2, ..., d_m >$ is used to perform clustering. Two physically close nodes have similar vectors. We use space-filling Hilbert curve [32, 33] to map each m-dimensional landmark vectors to a real number. The number is called the *Hilbert number* of a node denoted by $\mathcal{H}$. Nodes with closer $\mathcal{H}$ are physically closer to each other.

We choose the Hilbert curve based dimension reduction method because it outperforms the other dimension reduction methods under most circumstances [34], and it has been widely used in many areas such as data search, resource discovery and data retrieval [15, 35–37].

The HCO architecture builds a topology-aware cycloid architecture on a Grid. Specifically, each node i has an ID $(H_i, \mathcal{H}_i)$, where H_i is the consistent hash value of its IP address and $\mathcal{H}_i$ is its Hilbert number. When node i joins in HCO, it first generates its ID. Using the cycloid node joining algorithm, the node finds its place in the cycloid overlay and connects to its neighbors. Recall that for a cycloid ID

(id_s, id_l), id_s indicates node positions within a cluster and id_l differentiates clusters. By mapping $(H_i, \mathcal{H}_i)$ to (id_s, id_l) as shown in Figure 2, we see that physically close nodes which have the same $\mathcal{H}$ will be in the same cluster, and those having similar Hilbert number are in nearby clusters. Within a cluster, nodes are connected in the order of their H_i.

To build each node's routing table, HCO uses the proximity-neighbor selection technique [38]. That is, to choose a node for its routing table entry, a node first identifies all candidates that can become its neighbor in the entry based on the DHT policy. From these candidates, it selects the physically nearest node. As a result, the HCO architecture is constructed, in which the logical proximity abstraction derived from overlay matches the physical proximity information in reality.

Due to the dimension reduction, it is possible that a node is clustered with other nodes that are physically far away from itself or a node's closer node is clustered with other nodes. To deal with this problem, each node can check the correctness of the node clustering according to its neighbors' IP addresses and Round Trip Time (RTT) [39]. If a node's IP address does not match its neighbors' IP addresses and its RTTs with the neighbors are exceptionally longer than those with nodes in the neighboring clusters, the node transfers to its neighboring cluster. This process will finally lead the node to its right cluster. Due to uneven distribution of nodes in physical space, nodes may not be distributed in balance in the DHT ID space in the topology-aware cycloid. The work in [8] shows that the imbalance of node distribution in ID space does not affect the location efficiency in cycloid. Hence, it will not adversely affect the efficiency of resource discovery in HCO.

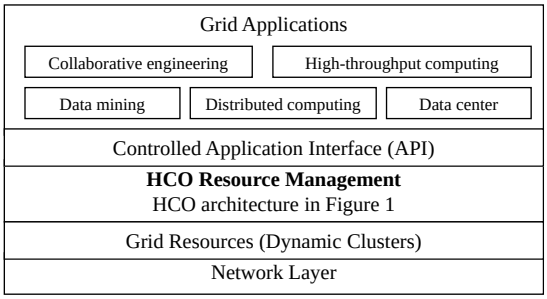


Fig. 3. Processing layers of using the HCO network for Grid resource discovery.

We show the processing layers of the HCO infrastructure in Figure 3. HCO enables scalable, efficient and robust resource discovery in large-scale distributed Grid applications. Examples of Grid applications include collaborative engineering, high-throughput computing, distributed modelling, data-mining, and data center systems. Applications apply the HCO resource discovery services using API calls. When a node reports its available resources to the system, it uses the `Insert(ID,object)` function. When a node needs a set of resources, it uses the `Lookup(ID)` function, which returns the locations of requested resources.

4 Locality-Preserving Grid Resource Management

We propose the locality-preserving resource clustering and discovery algorithms based on the HCO architecture. The idea is to map physically close functional resources to logically close nodes in order to satisfy specific application demands. Taking advantage of its hierarchical cluster structure, HCO uses the `Insert(ID,object)` function to group the descriptors of physically close nodes into the same cluster. Also, the logical distance between a node and a cluster on the HCO reflects the physical distance between the node and the resources whose descriptors are in the cluster. This facilitates a node to locate physically close resources by probing its nearby nodes in increasing proximity. Within a cluster, resource descriptors are further grouped according to resource functionality (i.e., resource type). This supports a node to discover resource based on its various functionality demands. Successful clustering leads to fast resource discovery for various Grid applications. HCO uses the `Lookup(ID)` function to discover multiple resources. Thus, HCO achieves program/data locality by supporting proximity-aware and multi-resource discovery. HCO also has strategies to deal with dynamism. The resource discovery efficiency is further enhanced by the cluster-token forwarding algorithm. Table 1 shows a summary of notations used in this paper.

4.1 Locality-Preserving Resource Clustering

An effective resource discovery algorithm locates resources across a wide area based on a list of predefined attributes. A node reports its available resources or requests for resources using a resource descriptor D_r, consisting of 4-tuple:

$$Resource\ Descriptor\ D_r =< RF, ID, RA, IP >,$$

where RF, ID and RA are the resource functionality, identifier and resource attribute. IP refers to the IP address of the resource owner or requester. For clarity, all node indices are omitted in the descriptors.

In a DHT overlay, the objects with the same ID are stored in the same node. Based on this object assignment policy, HCO computes the consistent hash value H_r of a resource r's resource functionality, and uses ID_r=$(H_r, \mathcal{H}_i)$ to represent the ID of resource r in node i. Each node applies `Insert(`ID_r`,` D_r`)` to periodically store the descriptors of its available resources in a node, which is called a *directory node*. The `Insert(`ID_r`,` D_r`)` function stores D_r to the node whose ID is closest to $ID_r = (H_r, \mathcal{H}_i)$. As shown in Figure 2, by mapping resource ID to HCO node ID, the directories of proximity-close resources are collected in one cluster, and the descriptors of the proximity-close resources with the same functionality are stored in the same directory node. Different nodes in a cluster are responsible for resources with different functionalities. Furthermore, resources in the directories stored in nearby clusters are located in physically close nodes.

We use *a directory node's resource* to represent the resource r whose D_r is stored in the directory node. The logical distances between node i and a number of directory nodes represent the physical distances between node i and the directory nodes' resources. Therefore, if a node has resource options in a number of directory nodes, it should choose the resource in the logically closest directory node in the overlay. Theorem 4.1 shows this feature of HCO.

Theorem 4.1: Assuming that node Hilbert number $\mathcal{H}$ can accurately reflects node proximity, if nodes j and k are directory nodes of the a resource requested by node i, and $ID_i \leqslant ID_j < ID_k$ or $ID_i \geqslant ID_j > ID_k$, then directory node j's resources are physically closer to node i than directory node k's resources.

Proof: With the assumption, if nodes j and k are directory nodes of one resource functionality, nodes j and k must be in different clusters. In HCO, the logical proximity abstraction derived from overlay matches the physical proximity information in reality. Therefore, if $ID_i \leqslant ID_j < ID_k$ or

$ID_i \geqslant ID_j > ID_k$, node j is physically closer to node i than node k. A node reports its resource descriptors to a node in its cluster, so directory node j's resources are physically closer to node i than directory node k's resources. □

Resource attributes (RA) are always expressed by strings or numbers [40]. For example, a query can be: "*OS=Unix and 800MHz≤CPU≤1000MHz and 512GB≤Memory≤2GB.*" The RA for the OS is a string that has a limited number of expressions, while the RA for CPU and Memory are numbers. Number-based RA have continuous numerical values. Thus, it is important for a resource discovery scheme to deal with range queries for resources in a RA range. Since the descriptors of one resource functionality in a cluster are gathered in one node, the node orders all the descriptors based on the numerical values of the resources. Assume a node keeps a directory with l entries. Then, it stores a descriptor with numerical value v to the $(v\%l)^{th}$ directory entry. When the node receives a resource range query with value $\geq x$, it searches the descriptors in the entries with index $\geq (x\%l)$. If the query range is $\leq x$, the node searches the descriptors in the entries with index $\leq (x\%l)$. If the query range is $x_1 \leq x \leq x_2$, the node searches the descriptors in the entries with index between $(x_1\%l)$ and $(x_2\%l)$. The node then returns the matching descriptors to the requester.

If a resource has a large number of string RA, a directory node keeps a list of unified strings for the RA. For instance, "Mem" and "memory" are unified to "Memory". For a resource descriptor, the node maps the resource RA in the unified list, calculates the consistent hash value H of the unified string, and stores the descriptor in the $(H\%l)^{th}$ entry in its directory. Later on, when the node receives a string-based query, it maps the string to the unified list and calculates the consistent hash value of the string, say h. The consistent hash function produces the same hash values for the same strings. The node then searches the descriptors in the h^{th} entry and returns the satisfying descriptors to the requester.

In the ideal case, the number of resource functionalities n_1 equals the number of nodes in a cluster n_2 and each node in the cluster is responsible for the directories of one resource functionality. When $n_1 > n_2$, a node in the cluster may store directories of more than one resource functionality. In this case, such a node maintains one directory for each resource functionality for ease of search. When $n_1 < n_2$, some nodes in the cluster do not act as directory nodes. In this case, a directory node can move part of its directories to non-directory nodes and keeps indexes to those nodes. Thus, it can move part of its load to other nodes for load balancing.

The load balancing algorithm in [20] can be further adopted to achieve more balanced descriptor distribution between the directory nodes. Since this is not the focus of this paper, we do not present the details of the load balancing algorithm.

4.2 Locality-Preserving Resource Discovery

When node i queries for multiple resources, it sends a request `Lookup`$(H_r, \mathcal{H}_i)$ for each resource r. Each request is forwarded to its directory node in node i's cluster. As shown in Figure 2, the request will arrive at the node whose ID $(H_j, \mathcal{H}_j)$ is closest to $(H_r, \mathcal{H}_i)$. Recall that `Insert(`ID_r`,` D_r`)` function stores D_r to the node whose ID is closest to $ID_r = (H_r, \mathcal{H}_i)$. Also, $\mathcal{H}_i$ represents the physical proximity of a resource requester or provider. Therefore, the requester can discover the resources within close proximity to itself.

If the directory node has no requested descriptor, it probes nodes in nearby clusters. Theorem 4.1 indicates that the resources of directory nodes in closer clusters are physically closer to the requester. Hence, a node should probe its logically close neighbors in order to locate physically close resources.

We present the successor and predecessor clusters of node j's cluster as *sucCluster(j)* and *preCluster(j)*, respectively. First, a node probes the directory nodes in these clusters simultaneously. Then, it probes the directory nodes in *sucCluster(sucCluster(j))* and *preCluster(preCluster(j))*. This process is repeated until the desired resource descriptors are found. However, such sequential probing is not robust enough to handle dynamism where nodes continually join and leave the system.

We develop the *proximity-aware randomized probing algorithm (PRP)* based on the algorithm in [20] to resolve the problem. In the PRP algorithm, a node first applies sequential probing. If no response is received during a predefined time period, the node randomly chooses two nodes in an increasing range of proximity and repeats the probing process. Since resource descriptors are allocated to different nodes in a cluster based on resource functionality, the probed nodes should be the directory nodes of the requested resource.

In the resource clustering algorithm, the `Insert(`ID_r`,` D_r`)` function stores D_r to the node whose ID is closest to $ID_r = (H_r, \mathcal{H}_i)$. Recall in cycloid ID (id_s, id_l), id_l differentiates clusters and id_s differentiates nodes in a cluster. Hence, $\mathcal{H}_i$ determines the cluster and H_r determines the node in the cluster where D_r should be stored. Recall a node's ID is $(H_i, \mathcal{H}_i)$. Thus, in each cluster, the directory node of one resource functionality has the cyclic ID H_i closest to H_r. Therefore, for directory node with ID $(H_i, \mathcal{H}_i)$ (i.e., probing node) of one resource functionality r in a cluster, the directory node of r in another cluster (i.e., probed node) must have the cyclic ID closest to H_i in that cluster. Consequently, the probing node i can reach the directory nodes of the requested resource by targeting an ID composed of its $id_s = H_i$ and a randomized id_l chosen in an increasing proximity.

The HCO scheme can also be applied to two-level hierarchical DHTs [41, 42] and other hierarchical overlays such as dBCube [43] and Kcube [44–46], where clusters are connected by a DHT structure based on de Bruijn graph or Kautz digraph. HCO is also applicable to variants of Chord [5] and Pastry [6], i.e., a hierarchical DHT based on Chord or Pastry. Accordingly, in these hierarchical structures, each cluster in the lower-layer is responsible for storing resource directories of physically close nodes. Nodes in a cluster are responsible for directories of different resources. A node generates its ID including its Hilbert number $\mathcal{H}_i$ using the same way in HCO. Similar to HCO, for a node with ID $(H_i, \mathcal{H}_i)$, $\mathcal{H}_i$ determines which cluster it locates and H_i determines the node's location in the cluster. The directory of resource with ID $(H_r, \mathcal{H}_i)$ is stored in a node in the same way as HCO. For a message for storing resource directories and discovering resources, it first is forwarded to the cluster with ID $\mathcal{H}_i$ through the routing in the upper-layer, and then is forwarded to the directory node through the lower-layer routing within a cluster. The design of HCO needs to be tailored to the specific designs of other systems. Due to the page limit, we do not discuss more details here.

4.3 An Example of the HCO Algorithms

Figure 4 shows an example of using HCO for locality-preserving resource clustering and discovery. Based on the ID determination policy in HCO, nodes $<a, b, c, d>$ generate

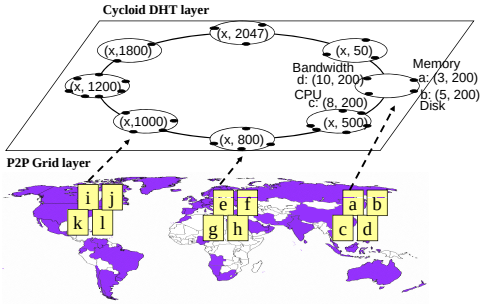


Fig. 4. Example use of the HCO network for global-scale Grid resource clustering and discovery. Grid resource clusters: $< a, b, c, d >$, $< e, f, g, h >$ and $< i, j, k, l >$ are created and managed by overlay nodes at three cycles in the cycloid hierarchy. Only partial connecting edges of the HCO overlay are shown.

their IDs=(x,200) (x denotes an arbitrary number), nodes $<e, f, g, h>$ generate their IDs=(x,800) and nodes $<i, j, k, l>$ generate their IDs=(x,1000). Thus, each of the physically close node groups $<a, b, c, d>$, $<e, f, g, h>$ and $<i, j, k, l>$ constitutes a cluster as indicated by the arrows in the figure. Within each cluster, all nodes generate resource ID_r for different resources and store the resource descriptors by `Insert(`ID_r`,` D_r`)`. Consequently, the resource descriptors of all nodes in the cluster are distributed among the nodes based on resource functionality. Assume H(Memory)=3, H(Disk)=5, H(CPU)=8 and H(Bandwidth)=10. In the cluster of $<a, b, c, d>$, all nodes generate IDs for their memory resource ID_r=$(3, 200)$, disk resource ID_r=$(5, 200)$, CPU resource ID_r=$(8, 200)$ and bandwidth resource ID_r=$(10, 200)$. By the `Insert(`ID_r`,` D_r`)` function, the nodes store their memory descriptors in node a, disk descriptors in node b, CPU descriptors in node c, and bandwidth descriptors in node d. Such locality-preserving overlay architecture construction and resource clustering facilitate nodes to discover various resources that are physically close to themselves.

Algorithm 1: Pseudo-code for node i's operations in HCO.

//clustering resource descriptors in local cluster
Generates consistent hash value of its resources: $(H_{r_1}, ..., H_{r_m})$
Generates $(ID_{r_1}, ..., ID_{r_m})$, $ID_{r_{\tilde{m}}}$=$(H_{r_{\tilde{m}}}, \mathcal{H}_i)$, $(1 \leqslant \tilde{m} \leqslant m)$
Applies DHT function `Insert(`$ID_{r_{\tilde{m}}}$`,` $D_{r_{\tilde{m}}}$`)`, $(1 \leqslant \tilde{m} \leqslant m)$

//requesting resource
Generates consistent hash value of the resource: $(H_{r_1}, ..., H_{r_m})$
Generates $(ID_{r_1}, ..., ID_{r_m})$, $ID_{r_{\tilde{m}}}$=$(H_{r_{\tilde{m}}}, \mathcal{H}_i)$, $(1 \leqslant \tilde{m} \leqslant m)$
Applies DHT function `lookup(`$ID_{r_{\tilde{m}}}$`,` $D_{r_{\tilde{m}}}$`)`
to get the D_r= $< RF, ID, RA, IP >$
Asks resources from resource owners

//processing resource request
Checks its directory for requested resources
while has not found the matching D_r **do**
  Applies the PRP algorithm to search from neighboring directory nodes of the requested resource
Returns the D_r

The requests for memory and disk resources from nodes $< a, b, c, d >$ are also forwarded to nodes a and b, respectively. Specifically, when a node in a cluster needs memory and disk resources, it first generates the IDs for the memory resource ID_r=$(3, 200)$ and for the disk resource ID_r=$(5, 200)$ and then uses `Lookup(3,200)` for memory and `Lookup(5,200)` for disk space. Using the DHT routing algorithm, the requests are forwarded to nodes a and b, respectively. These nodes check their own directories for the descriptors of the requested resource. If they cannot find the descriptors, they probe nodes in other clusters using the PRP algorithm. For example, node b (5,200) probes nodes by targeting (5,199) and (5,201). If it does not receive a reply within a predefined time period, it randomly generates two cubical indices within 100 proximity range. Suppose the two randomized numbers are 150 and 250, the node probes nodes by targeting (5,150) and (5,250). If the requested resource is still not found, the node increases the proximity range and repeats the same process until it finds the requested resource. Algorithm 1 shows the pseudo-code of resource clustering and discovery performed by a node in the HCO network.

4.4 Dynamism-Resilient Resource Management

In addition to exploiting the physical proximity of the network nodes to minimize operation cost, an effective resource management scheme should also work for Grids in a dynamic environment. HCO uses the cycloid self-organization mechanism to cope with these problems. Specifically, nodes transfer descriptors when joining or leaving the system.

When node i joins in the system, it reports its resources via `Insert(`$(H_r, \mathcal{H}_i)$`,` D_r`)`, and receives the descriptors in its responsible ID region from its neighbors. When a node departs from the system, it transfers descriptors to its neighbors. For example, if node $(2, 200)$ joins the system in Figure 4, then the descriptors in the range $(0, 200)$ and $(2, 200)$ are transferred from node $(3, 200)$ to node $(2, 200)$. If node $(3, 200)$ leaves, it transfers its descriptors to node $(10, 200)$ or $(5, 200)$ based on the ID closeness. If node $(3, 200)$ is the only node in its cluster, it transfers its descriptors to its closest node in its closest cluster.

HCO resorts to periodical resource reporting to avoid useful descriptors from being lost in the clustering and discovery process. If a directory node has failed, its resource descriptors are lost. In this case, the resource requests will arrive at the node that is the new directory node of the resource. Since the directory node does not have resource descriptors satisfying the requests, it keeps the requests in its request queue. In the next periodical resource reporting, the lost resource descriptors will be reported to the new directory node. Then, the queued requests can be resolved. For example, in Figure 4, if node a fails, all of its resource descriptors are lost. Also, the requests for the memory resource are forwarded to node b based on the DHT routing algorithm. In the next resource reporting period, by the `Insert()` function, the memory descriptors are forwarded to the new directory node of memory, node b. To prevent the descriptor space from being flooded with outdated descriptors, the directory nodes discard outdated resource descriptors periodically. Consequently, instead of relying on specific nodes for resource descriptors, HCO always stores a resource descriptor in a directory node, and the `Lookup(`$H_r, \mathcal{H}_i$`)` requests can always be forwarded to the node.

4.5 Cluster-Token Forwarding Algorithm

We introduce a cluster-token forwarding algorithm to further enhance the efficiency of the HCO scheme. Like most multi-resource management approaches, HCO uses m lookups for a query of m resources. Based on the cycloid routing algorithm, all lookup messages first are routed within a cluster sequentially. Thus, rather than using m lookups, a node can combine the m lookups into one lookup message

to be sequentially routed within a cluster. Moreover, since a node with m available resources needs m `Insert()` messages for resource clustering which are routed in the same manner as the `Lookup()` messages, the two kinds of messages can be integrated. Furthermore, since the messages of all nodes in a cluster are routed in the same manner, and the nodes need to report their available resources periodically, the messages for resource clustering and discovery of all the nodes can be combined.

Based on this observation, the cluster-token forwarding algorithm accumulates the messages of available resources and resource requests of all nodes in one cluster. In a nutshell, the primary node in each cluster periodically generates a token which circulates along its cluster. Each node receiving the token inserts the resource descriptors of its available resources and resource requests into the token, absorbs the descriptors of available resource and resolves the resource requests in the token that are in its responsibility.

For example, if primary node i needs multiple resources represented by $r_1, r_2, ..., r_{m1}$, and it has available resources represented by $\delta r_1, \delta r_2, ..., \delta r_{m2}$, it generates the IDs of the resources $ID_{r_1}, ..., ID_{r_{m1}}$ and $ID_{\delta r_1}, ..., ID_{\delta r_{m2}}$, in which

$$ID_{r_{\tilde{m}}} = (H_{r_{\tilde{m}}}, \mathcal{H}_i)(1 \leqslant \tilde{m} \leqslant m1 + m2).$$

The resource descriptors are ordered by H_r in the form of

$$< D_1, D_2, \cdots, D_{m1+m2} > .$$

Next, node i sends the token to its successor j.

Based on the HCO resource clustering algorithm, a node is the directory node of the resources whose H_r satisfies

$$ID_{pre.cyc} \leqslant H_r \leqslant ID_{suc.cyc},$$

where $ID_{pre.cyc}$ and $ID_{suc.cyc}$ represent the cyclic index of the node's predecessor and successor, respectively. Therefore, in order to avoid unnecessary checking, node j only needs to check the D_r in the token satisfying this condition. For those D_r of requests, if node j has resource descriptors of the required resources, it sends the resource descriptors to the requesters, and removes the D_r from the token. For those D_r of available resources, node j absorbs the D_r. Afterwards, it inserts its own D_r of available resources and requests into the token, and forwards the token to its successor. This process is repeated until the primary node receives the token back, which means that the token has completed one circulation.

At this time, the token has no D_r of available resources and the D_r left are for unsolved resource requests. The primary node then uses PRP to forward the token to another cluster, where this process is repeated until the token is empty, i.e., all requested resources are discovered. Consequently, in the algorithm, only one message is generated periodically. Combining a number of messages into a single message for forwarding within a cluster and between clusters significantly reduces cost.

Algorithm 2 shows the pseudo-code for the cluster-token forwarding algorithm in HCO. This algorithm is suitable to the situation when many messages travel most of the nodes in the cluster. For the resource discovery operation, it is possible that only a few nodes in a cluster request resources with few functionalities in resource discovery. In this case, circulating the combined requests may generate more overhead. For example, if there is only one node sending out a resource request with one functionality, then forwarding its request directly to the destination directory node costs less than circulating it using the algorithm. However, since every node in a cluster needs to report its available resources periodically, combining all resource reporting messages and resource requests can reduce the total cost for both resource reporting and querying.

Algorithm 2: Pseudo-code for cluster-token forwarding algorithm.

//executed by primary node i periodically
Generates a token
if it has a request for multi-resources or available resources **then** {
  Generates the ID of the resources:$(H_r, \mathcal{H}_i)$
  Inserts each D_r into the token in the ascending order of H_r }
Sends the token to $suc(i)$
//after primary node i receives a token
if receives a token **then**
  Forwards the token to a node in another cluster using PRP

// node i receives a token $< D_1, D_2, \cdots, D_{m1+m2} >$
// node i handles its responsible D_r in the token
Finds the D_r group $\mathcal{D}$
$=\{\forall\ D_r =< RF, ID, RA, IP > |(ID_{pre.cyc} \leqslant H_r \leqslant ID_{suc.cyc})\}$
for each $D_r \in \mathcal{D}$ **do**
  if D_r is for requested resource **then** {
    // handles D_r of resource requests
    Checks its directory for r
    if it has the requested resource r **then** {
      Replies the requester with IP
      Removes the D_r from the token }}
  else { *// handles D_r of available resources*
    Transfers D_r from the token to its directory
    Removes the D_r from the token }
//adds its D_r into the token
if has a multi-resource request or has available resources **then** {
  Generates the ID of the resources: $(H_r, \mathcal{H}_i)$
  Inserts D_r to the token in the ascending order of H_r }
Sends the token to $suc(i)$

Theorem 4.2: In a d-dimensional HCO network, the cluster-token forwarding algorithm can always reduce the number of contacted nodes for a resource query with m $(m \geq 2)$ resources.

Proof: A query for one resource needs d contacted nodes in the requester's cluster. It needs d nodes for message forwarding in one probing [8] in the average case. Therefore, for a resource query with m resource, $m(d+d\cdot 2^d)$ contacted nodes are needed without the cluster-token forwarding algorithm. With the algorithm, d contacted nodes are needed for a token circulation, $d\cdot 2^d$ contacted nodes for message routing, and $d\cdot 2^d$ contacted nodes for token circulation in all clusters. Thus, the total number of contacted nodes needed is $d + d \cdot 2^d + d \cdot 2^d$. Therefore, as long as $m > \frac{d+d\cdot 2^d+d\cdot 2^d}{d+d\cdot 2^d} = 1 + \frac{d\cdot 2^d}{d+d\cdot 2^d}$, the contacted nodes without the algorithm is more than that with the algorithm. □

4.6 Deadline-driven Resource Management

Since some Grid systems used for complex scientific applications are time critical and must comply to strict QoS rules, significant challenges still remain to support distributed applications that easily suffer from latencies. They require discovering desired resources in a timely fashion for resource requesters. To address this problem, HCO uses deadline-driven resource allocation and message forwarding algorithms in a distributed manner to ensure that the requested resources are provided in time. We use deadline to denote the time instant that a resource is needed by a requester. The basic idea of the algorithms is to (1) process urgent requests immediately without waiting for the next periodical resource allocation, (2) process resource requests in the ascending order of their

deadlines, and (3) forward messages in routing in the ascending order of their deadlines in order to make sure that requests arrive their destinations in time.

In addition to the current available resources, HCO requires each node to also report its available resources in future time slots. On the other hand, resource requesters can ask for resources they need instantly, and they can also reserve resources for a specific future time slot.

Each directory node has a pair of sorted available resource list (ARL) and requested resource list (RRL) for resources whose IDs are located in its responsible region. An ARL is used to store the descriptors of available resources and their available time instants, and a RRL is used to store the descriptors of requested resources and their deadlines, as shown in Table 2. We let T to denote the resource available time or deadline. To distinguish resource requests and reports for available resources, we use D_r=$< RF, ID, RA, IP>$ to represent the former and use $D_{\delta r}$=$<RF, ID, \delta RA, IP>$ to represent the latter. The resource information of an available resource and a requested resource is represented in the form of $<D_{\delta r}, T>$ and $<D_r, T>$, respectively. Both ARL and RRL are sorted in ascending order of the time T. For the same T, they are ordered in the manner introduced in Section 4.1 for range querying. Resource allocation is executed between a pair of ARL and RRL to allocate available resources to resource requesters in a top-down fashion. More precisely, each directory node runs Algorithm 3.

Algorithm 3: Pseudo-code for resource allocation performed by a directory node.

```
Sort ARL and RRL in ascending order of the time T
for each item j <D_rj, T_j> in RRL do
  for each item i <D_δri, T_i> in ARL do
    if T_i ≤ T_j then{
      if RA is a value && δRA ≥ RA ||
         RA is a string && H(δRA) = H(RA) then{
        Inform IP_j of the resource provider D_δri
        Remove <D_rj, T_j> from RRL
        if RA is a value && (δRA − RA > 0) then{
          Replace δRA in D_δri with (δRA − RA)
          Move <D_δri, T_i> to the right position in the directory}
        else
          Remove <D_δri, T_i> from ARL}
      }
```

The resource providers and requesters periodically report their resource directories to directory nodes, which allocate available resources to requesters periodically. If a directory node cannot find a resource with available time slot earlier than a resource request's deadline in the resource allocation, this means there are no available resources in the system that can meet the deadline. Then, the directory node sends a negative response to the requester. To handle urgent resource requests, we define a parameterized threshold E for T to determine if a request is urgent or not. For an urgent request, a directory node immediately handles it without waiting for its next periodic resource allocation. The directory node then replies to the resource requester immediately about the resource locations.

In addition to reserving resources beforehand in order to guarantee the resource availability, it is important to ensure that the resource management operation itself is performed efficiently. Specifically, the messages for resource requests and for resource location notification should arrive at their destinations without delay. Deadline-driven message forwarding algorithm is used for this purpose. In the algorithm, a resource request is forwarded with deadline information. When a node receives multiple messages, it forwards the messages based on their deadlines. A message with earlier deadline has higher priority to be forwarded.

TABLE 2
Sorted available and requested resource lists.

Directory of resource information in a node	
Available resource list	Requested resource list
$< D_{\delta r_1}, T_1 >$	$< D_{r_1}, T_1 >$
...	...
$< D_{\delta r_m}, T_m >$	$< D_{r_n}, T_n >$

5 Performance Evaluation

We designed and implemented a simulator for evaluation of the HCO scheme. We compared the performance of HCO with MAAN [11], Mercury [10], and SWORD [12]. To be comparable, we used Chord for attribute hub in Mercury and SWORD. The experimental results show advantages in using HCO over the competing overlays for the same purpose.

The works in [47, 48] use bounded Pareto distribution for node capacity. Also, the size distribution of process CPU and memory requirements fits a bounded Pareto quite accurately [49–53]. In addition, the empirical observations by [52] show that process lifetime distributions in Unix systems can be approached with a Bounded Pareto distribution. Therefore, we used the Bounded Pareto distribution function to generate the resource amount owned and requested by a node. This distribution reflects the real world where there are available resources that vary by different orders of magnitude.

The number of nodes in the system was set to 4096. We assumed that there are 11 types of resources. The dimension of the cycloid simulated was set to 11. In each experiment, we first let each node report its available resources, and then randomly chose nodes to generate 1000 requests. The number of resources in a request or the number of available resources per node was varied from 1 to 5 with step size of 1, unless otherwise specified. The values of the settings are randomly chosen from a reasonable range. Different values will not change the relative performance differences between the methods. We use HCO/o to represent HCO without the cluster-token forwarding algorithm.

We randomly chose 15 landmarks with a constraint that the distance between each pair of landmarks is at least 4 hops. The setting values are our empirical values for appropriate node clustering. More landmarks lead to more accurate nodes clustering but higher overhead, and vice versa [31, 32]. We used transit-stub topologies generated by GT-ITM [54]: "ts5k-large" and "ts5k-small" with approximately 5000 nodes each. "ts5k-large" is used to represent a Grid consisting of nodes from several big stub domains, while "ts5k-small" represents a Grid consisting of nodes scattered throughout the entire Internet and only few nodes from the same edge network join the overlay. We evaluate the effectiveness of the resource management approaches using the following metrics:

(1) *Cumulative distribution function (CDF) of the percentage of discovered resources.* This reflects the effectiveness of a resource discovery algorithm to discover requested resources physically close to requesters.

(2) *Physical/logical communication cost.* The communication cost is directly related with message size and routing path length (logical/physical). We use the product of these two factors to represent the communication cost. It is assumed that the size of a resource request message or

response message is 1 unit.

(3) *Overlay maintenance cost.* To maintain the overlay, each node periodically executes stabilization by probing its neighbors to ensure that they are alive. We use the total number of probing messages of all nodes in one stabilization operation to demonstrate the overlay maintenance cost. This metric represents the overhead to maintain the DHT resource management architecture.

(4) *Directory size.* This affects balanced distribution of workloads caused by resource descriptor maintenance and resource request processing in resource management.

(5) *Resource request success rate.* This is the percent of resource requests that reach their directory nodes successfully. This metric represents the capability of a resource management scheme to deal with dynamism.

(6) *Number of probed nodes.* Because of the node and resource dynamism, a request may arrive at a node with no descriptor of the requested resource. Then, its nearby nodes are probed for the resource. This metric is defined as the number of the probed nodes. It reflects the impact of dynamism on the efficiency of a resource management scheme.

(7) *Number of overdue request responses.* This is the number of responses received after the request deadlines by the requesters. This metric shows the effectiveness of the proposed deadline-driven resource management algorithm in enhancing the speed of resource discovery.

5.1 Locality-preserving Resource Discovery

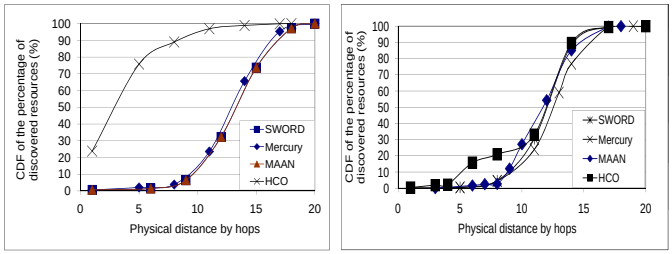
This section shows the effectiveness of HCO in locality-preserving resource management, in which resources physically close to requesters are discovered. In this experiment, we randomly generated 5000 resource requests, and recorded the distance between the resource provider and requester of each request. Figure 5(a) and (b) show the CDF of the percentage of allocated resources against the physical hop distance in "ts5k-large" and "ts5k-small", respectively. We can see that in "ts5k-large," HCO is able to locate 97% of total resource requested within 11 hops, while others locate only about 15% within 10 hops. Almost all resources are located within 15 hops from requesters in HCO, while 19 hops in other resource management methods. The results show that HCO can locate most resources within short distances from requesters, but others locate most resource in long distances. From Figure 5(b), we have the same observations, although the performance difference between approaches is not so significant. The results confirm the unique locality-preserving feature of HCO to enable users to locate physically close resources. Communication cost also plays an important role in resource management efficiency.

5.2 Cost in Resource Management

This section shows the low cost of HCO in resource management. In HCO, nodes only need to communicate with their physically close nodes for resource clustering and discovery. Also, nodes only need to perform cluster-wide communication for resource clustering and discovery rather than system-wide communication as in other schemes. We tested the physical and logical communication cost in "ts5k-large" and "ts5k-small". As the results in "ts5k-small" are similar to those in "ts5k-large", we do not include the results due to space limit.

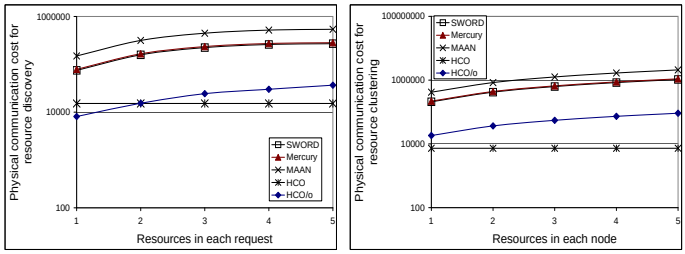
Figure 6(a) and (b) plot the physical communication cost for resource discovery and clustering versus the number of resources in a query, respectively. In order to separate the cost for resource discovery and clustering, we used the cluster-token forwarding algorithm to combine the resource requests and reporting messages, respectively. Actually, these two kinds of messages should be combined together in message circulation. In the experiment, searching stops once a requested resource is discovered. From these figures, we can see that the cost of each scheme increases with the number of resources in a query. The cost of MANN grows dramatically faster than others, while HCO and HCO/o only have a marginal increase. Recall that MAAN needs two messages for each resource discovery and resource reporting, one is for resource type and the other for resource attribute. This generates much higher communication cost. The results illustrate that HCO and HCO/o incur much lower physical communication cost than others by arranging nodes to contact their physically close nodes. A requester can resolve most of its resource requests within its cluster, and it only needs to contact physically close nodes for the resource, resulting in much lower cost. SWORD and Mercury do not consider the proximity in the resource management process, so they generate much higher cost than HCO and HCO/o. The results also show that HCO/o leads to higher cost than HCO when the number of resources in one request is larger than 1 in the resource discovery phase and in all cases in the resource clustering phase. This implies that the cluster-token forwarding algorithm is effective in reducing communication cost by combing a number of messages into one in routing. It is intriguing to see that the cost of HCO is higher than HCO/o when the number of resources in each request equals to 1 in the resource discovery phase, but the cost of HCO is lower than HCO/o in this case in the resource clustering phase. This is due to the reason that when there are only a few requests in resource discovery, HCO/o directly route the requests to their destination, while HCO still circulates a token along an entire cluster hop by hop, generating more visited nodes than HCO/o. In the resource clustering phase, each node in a cluster needs to report its available resources, then the cluster-token forwarding algorithm shows its advantage by combining the messages from all nodes in a cluster together in routing.

In a resource discovery method, the number of hops required to forward a message is determined by the time complexity (i.e., lookup path length) of its underlying overlay. The time complexity of cycloid is $O(d)$ [8] while that of Chord is $O(\log n)$ [5], where n is the number of nodes in the system. Figure 7(a) and (b) plot the logical communication cost versus the number of resources in a query for resource discovery and clustering, respectively. We can observe that MAAN renders tremendously higher logical communication cost than SWORD and HCO. MAAN needs two lookups for each request, so it produces twice the number of visited nodes. HCO incurs less cost than Mercury and SWORD. In Mercury and SWORD, request forwarding is performed on a system-wide scope involving all nodes in the system. The average lookup path length is $\log n/2 = 6$ in Chord. In contrast, due to HCO's locality-preserving feature, a node can locate resources in its cluster with smaller scope. Since the maximum number of nodes in a cluster in HCO is 11, the average maximum lookup path length is $\log d/2 = 5.5$. Therefore, HCO generates shorter request path length. A request with m resources needs m lookups, which amplifies the difference of lookup cost between Mercury/SWORD and HCO. With the cluster-token forwarding algorithm, HCO integrates messages within a cluster into one message, leading to less visited nodes and a lower communication cost. We can also see that the cost of HCO is higher than HCO/o when the number



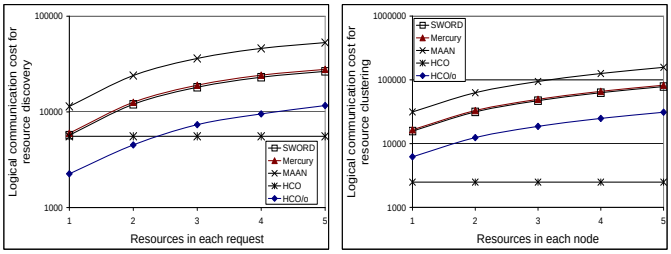
(a) "ts5k-large" (b) "ts5k-small"

Fig. 5. Locality-preserving resource discovery.



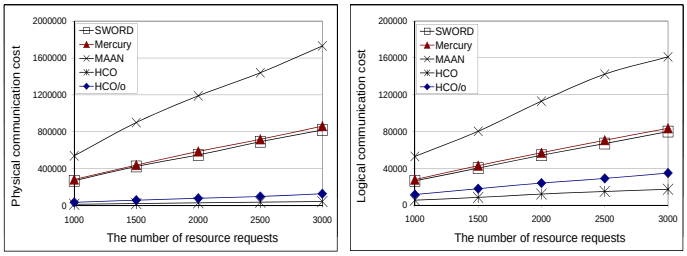
(a) Resource discovery (b) Resource clustering

Fig. 6. Physical communication cost.



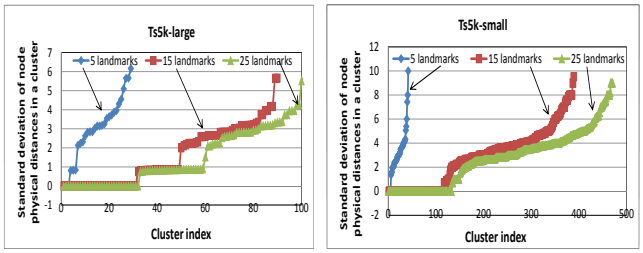
(a) Resource discovery (b) Resource clustering

Fig. 7. Logical communication cost.



(a) Physical communication cost (b) Logical communication cost

Fig. 8. Communication cost for resource discovery.



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

Fig. 9. Effectiveness of landmark clustering.

of resources in each request is 1 and 2 due to the same reason as Figure 6(a). In Figure 7(b), HCO has lower cost than HCO/o in all cases since all nodes in a cluster need to report available resources and combining their messages saves the cost. These experimental results are consistent with Theorem 4.2, and confirm the effectiveness of the cluster-token forwarding algorithm.

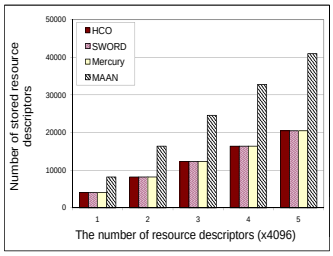
Figure 8(a) demonstrates the total physical communication cost versus the number of resource requests. In this experiment, each request has five resources and the messages for resource requesting and reporting are combined together. The figure shows that MAAN generates the highest physical communication cost. Mercury and SWORD produce approximately the same cost. HCO and HCO/o produce significantly lower cost than others, with HCO producing the least cost. The results are consistent with Figure 6(a) due to the same reasons. MAAN generates many more messages for the same number of resource requests since it needs two messages for one query. MAAN, Mercury and SWORD produce higher communication cost than HCO due to two reasons. First, they search resources in a system-wide scope while HCO searches resources in a cluster. Second, they do not take node proximity into account, thus nodes may communicate with nodes that are physically far away, while HCO considers proximity. In HCO and HCO/o, nodes communicate with physically close nodes for resource discovery, leading to much less physical communication cost. Also, by employing the cluster-token forwarding algorithm that combines both resource reporting and requesting messages together in message transmissions, HCO reduces the logical communication cost of HCO/o.

In order to see the effect of the number of messages and the searching scope on the physical communication cost, we measured the total logical communication cost and plot the results in Figure 8(b) versus the number of resource requests. The figure shows that MAAN generates the highest logical communication cost. Mercury and SWORD produce approximately the same cost. HCO and HCO/o produce significantly lower cost than others, with HCO producing the least cost. The results are consistent with Figure 7(a) due to the same reasons except the proximity-consideration. Comparing Figure 8(b) with Figure 8(a), we find that HCO and HCO/o reduce more physical communication cost of other schemes than the logical communication cost. This is because HCO and HCO/o further consider proximity while other schemes do not consider it.

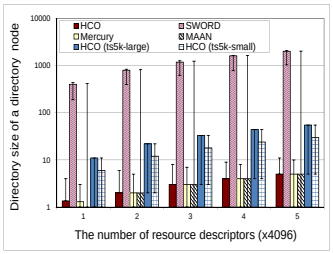
5.3 Overhead in Grid Resource Management

The resource descriptors need to be maintained in directory nodes. Resource descriptors of available resources are distributed among nodes for resource discovery. Figure 10(a) plots the total number of stored resource descriptors in the system versus the number of descriptors of available resources. The number of descriptors of available resources was varied from $4096 \times c$ ($c \in [1, 5]$). We can observe that MAAN generates twice the number of stored resource descriptors of other resource management schemes, and other schemes produce the same number of stored resource descriptors. For one resource descriptor of one available resource, HCO, SWORD and Mercury store one resource descriptor in the system. HCO stores resource descriptors based on resource attributes and resource owners' physical locations. SWORD stores resource descriptors based on resource attributes. Mercury keeps multiple DHTs and stores resource descriptors to the corresponding DHT based on resource values. MAAN stores two copies of resource descriptor in the system for each resource descriptor based on resource attributes and values. Consequently, it doubles the stored resource descriptors of other schemes. Therefore, MAAN needs much more cost for resource descriptor maintenance than other schemes.

Figure 9 (a) and (b) show the standard deviation of node physical distances in each cluster in "ts5k-large" and "ts5k-



(a) Total num of resource descriptors



(b) Directory size

Fig. 10. Resource descriptor maintenance cost.

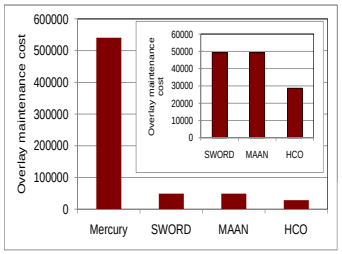


Fig. 11. Overlay maintenance cost.

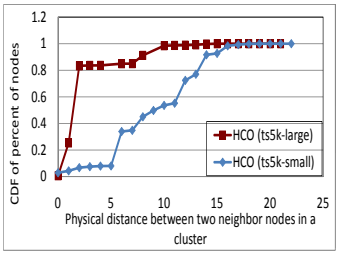


Fig. 12. Effectiveness of node clustering.

small", respectively. Here, a cluster means a group of nodes with the same Hilbert number. Suppose a cluster has m nodes, its standard deviation of node distances is calculated by $\sqrt{\sum_{i=1}^{m}(D_i - \overline{D})^2/m}$, where D is the physical distance between a pair of nodes and $\overline{D}$ is the average of D. In both figures, we see that 5 landmarks lead to high deviation values for most clusters. 25 landmarks and 15 landmarks generate many clusters with 0 and low deviation values. Also, 25 landmarks produce more clusters with lower deviations than 15 landmarks. The experiment results confirm that more landmarks generate more accurate node clustering, and vice versa.

Directory size is a metric of the balanced distribution of load. It is desirable to distribute the resource information among nodes evenly so that the information maintenance overhead, as well as the load for processing resource requests, can be distributed among nodes in order to avoid bottlenecks. We considered 1 to 5 resource descriptors for available resources per node, and measured the median, 1st and 99.9th percentiles of directory sizes of directory nodes.

Figure 10(b) plots the measured results versus the total number of resource descriptors. In the figure, HCO represents the HCO when nodes are evenly distributed in the geographical area. "HCO (ts5k-large)" and "HCO (ts5k-small)" respectively represent the HCO in the "ts5k-large" and "ts5k-small" topologies where nodes are not evenly distributed. In "ts5k-large", there are 90 clusters and the maximum, middle and minimum number of nodes in a cluster is 331, 6 and 1, respectively. In "ts5k-small", there are 470 clusters and the maximum, middle and minimum number of nodes in a cluster is 256, 3 and 1, respectively.

Three observations can be made from the figure. First, the median size of SWORD is much higher than others. SWORD clusters descriptors into 11 nodes responsible for the 11 resource types. Thus, each of the 11 nodes has large directory size while other nodes do not have workload, leading to serious load imbalance. This is confirmed by the experiment results showing that all other nodes have no descriptors. Second, MAAN exhibits significantly larger variance than others, and its 99.9 percentile is similar to SWORD's. MAAN stores a descriptor twice based on resource type and attribute respectively. Thus, like SWORD, MAAN assigns a much higher workload to the 11 nodes. Because the hash values of resource attributes are widely spread along the DHT ID space, the descriptors are also distributed among other nodes. As a result, some nodes have much fewer descriptors and the 11 nodes have much more descriptors, resulting in load imbalance. Third, Mercury and HCO produce much smaller median directory sizes and lower variance. Mercury uses a DHT for each resource, and classifies resource descriptors based on attribute in each DHT, which helps to distribute resource descriptors in balance. By taking advantage of the hierarchical structure of cycloid, HCO pools the descriptors in a cluster and allocates the descriptors to the cluster nodes based on resource type. We also see that "HCO (ts5k-large)" and "HCO (ts5k-small)" produce higher median, 1st and 99.9th percentile values than HCO. It is because in these two topologies, nodes are not evenly distributed in the geographical area. Thus, some clusters have much more nodes than others, leading to unbalanced resource directory distribution. We find that their 99.9th percentile values are still significantly lower than SWORD and MAAN. Therefore, Mercury and HCO can achieve a more balanced distribution of workload.

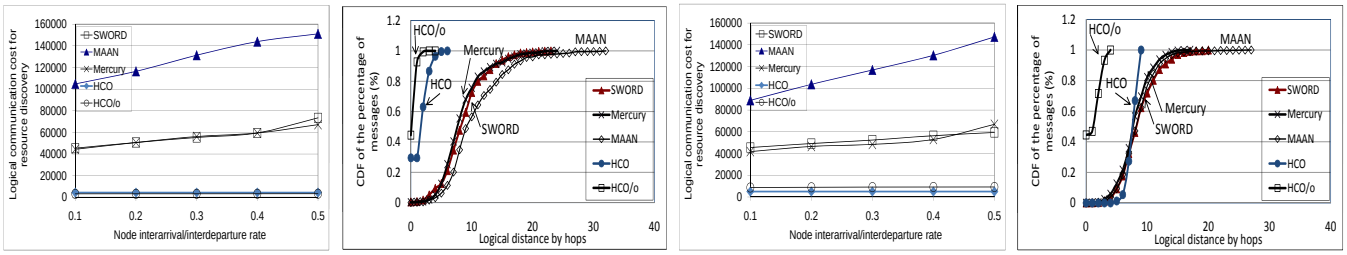
Figure 11 plots the overlay maintenance cost in different resource management schemes. The results show that Mercury generates a significantly higher maintenance cost than others. This implies that each node in Mercury maintains more neighbors than others. Recall that Mercury has multiple DHT overlays with each DHT overlay responsible for one resource. Therefore, a node has a routing table for each DHT, and has out-degree equals to the routing table size multiplied by the number of DHT overlays. Thus, each node needs to probe many more neighbors to maintain its outlinks in Mercury than others. The figure inside Figure 11 shows the overlay maintenance cost of MAAN, SWORD and HCO. We can see that HCO leads to much less overlay maintenance cost than MAAN and SWORD. This is because MAAN and SWORD are built on Chord with $\log n$ out-degree per node. HCO is built on cycloid with constant 7 out-degree. The results imply that HCO has a higher scalability with lower overlay maintenance cost than other resource management schemes.

5.4 Effectiveness of Node Clustering

Figure 12 shows the CDF of the percent of nodes versus the physical distance by hops between two neighbor nodes in a cluster in "ts5k-large" and "ts5k-small." In both topologies, the distance between two nodes ranges from 0-22 physical hops. We see that in "ts5k-large" where nodes are from several big sub domains, 83% neighbor nodes are within 2 physical hop distance, and only 1.5% neighbor nodes are beyond 10 hops. In "ts5k-small" where nodes are scattered throughout the network, 92% neighbor nodes are in 10 hops, and 7% neighbor nodes are beyond 10 hops. The results show that using the landmarks and Hilbert curve to calculate node proximity can cluster physically close nodes with high accuracy. Hence, HCO generates lower overhead than other resource discovery methods, since physically close node communicate each other in overlay maintenance and resource querying.

5.5 Performance in a Dynamic Environment

In this experiment, we ran each trial of the simulation for 20T simulated seconds, where T is a parameterized resource clustering period, which was set to 60 seconds. We ranged the node arrival/departure time rate from 0.1 to 0.5 with 0.1 step



(a) Resource request cost (d=11) (b) Percentage of messages vs. logical distances (d=11) (c) Resource request cost (d=9) (d) Percentage of messages vs. logical distances (d=9)

Fig. 13. Efficiency of resource management schemes in dynamism.

size, and generated 5000 resource requests randomly. For instance, there was one node join and one node departure/failure every 2.5 seconds at rate of 0.4. The resource join/departure rate was modelled by a Poisson process with a rate of 0.4 as in [8]; the resource type in requests is randomly distributed.

In this experiment, the nodes are randomly distributed. In DHTs, a node has a leaf set recording its predecessor(s) and successor(s). When a node departs from the system gracefully, it notifies the nodes in its leaf set. Since DHTs is churn-resilient with their stabilization mechanism. In order to test the churn-resilience of the schemes, the stabilization was turned off in the experiment. Figure 13(a) shows the logical communication cost of each scheme when the cycloid dimension $d = 11$. The cost is for both successful and unsuccessful resource requests. We observe that the cost of each scheme follows MAAN>SWORD≈Mercury>HCO/o≈HCO. In HCO and HCO/o, a node's request is forwarded to the directory node of the resource in the same cluster. In graceful node departures, the structure of each cluster is always maintained. Since the system has 4096 nodes and around $2^{11} = 2048$ clusters, the number of nodes in each cluster is small. Thus, a request always travels only a few nodes in one cluster before arriving at the directory node, leading to low cost. Since the routing path is not affected by the node joins and departures, the cost of HCO and HCO/o remain nearly constant.

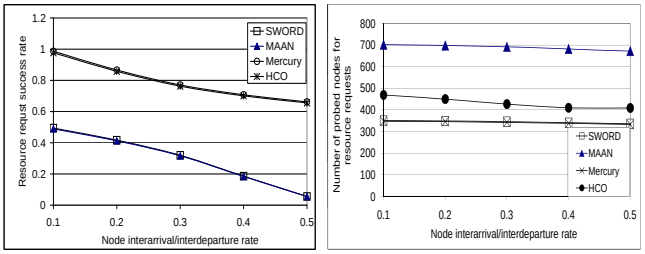
In contract, in MAAN, SWORD and Mercury, a request needs to travel in a system-wide scope through node routing table. Because the routing tables are not updated in node join and departures, a request needs to travel more hops than in a static environment. Fast node departures and joins lead to more outdated neighbors in node routing tables and hence longer path lengths. Thus, MAAN, SWORD and Mercury produce higher logical communication cost than HCO. Also, as the node interarrival/interdepature rate increases, their costs increase. MAAN has much higher cost than others because it sends two messages for one resource request.

We also see that HCO/o and HCO generate approximately the same cost. HCO combines the requests in a cluster into one message. When there are few nodes in a cluster (around 4096/2048=2), the message combination cannot reduce the cost. For example, in a cluster with two nodes, the circulation of a message takes two hops, while a message only takes one hop to arrive at the other node. At a result, HCO cannot reduce the cost of HCO/o.

Table 3 illustrates the number of request routing failures. It shows that the number of failures of MAAN, SWORD and Mercury increases as the node interarrival/interdeparture rate increases. When a node cannot find a neighbor to route a request, a routing failure occurs. Faster node joins and departures make more nodes in the routing tables outdated, leading to more routing failures. Since the leaf sets of nodes

TABLE 3
The number of request routing failures.

Rate	MAAN	SWORD	Mercury	HCO	HCO/o
0.1	0	0	7	0	0
0.2	53	11	15	0	0
0.3	98	43	57	0	0
0.4	157	152	152	0	0
0.5	261	176	208	0	0



(a) Request success rate (b) The num. of probed nodes

Fig. 14. Effectiveness of resource management schemes in dynamism.

are always updated and the routing in HCO is within a cluster most of the time, leading to 0 failures in both HCO and HCO/o.

To further investigate the effect of node dynamism on the logical communication cost, we depict the CDF of the percentage of messages versus the logical distances traveled by the messages in Figure 13(b). The figure shows that most messages travel short logical distances in HCO and HCO/o, but travel long logical distances in other schemes. Specifically, 99% of messages travel 2 hops in HCO/o, travel 5 hops in HCO, travel 22 hops in SWORD, 21 hops in Mercury, and 26 hops in MAAN. The results confirm the dynamism-resilience of HCO and HCO/o due to their smaller-scope searching within clusters. This saves the routing cost and mitigate the adverse influence of dynamism on routing as shown in Figure 13(a). On the contrary, in MAAN, SWORD and Mercury, requests must be forwarded through routing tables which cannot be updated without stabilization, leading to longer routing path lengths. Messages in HCO/o travel longer distances than HCO on average due to the same reason as explained in Figure 13(a).

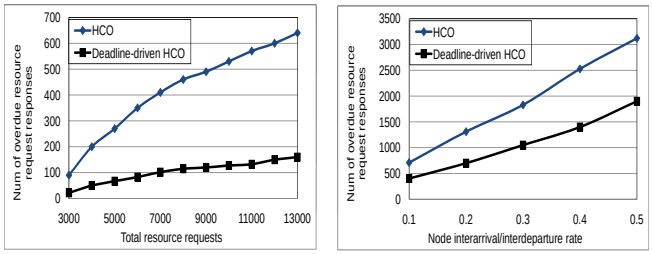
We set cycloid dimension $d = 9$ and repeat the same experiments in order to see the effect of node density in a cluster on the logical communication cost. With $d = 9$ and hence around $2^6 = 512$ clusters, the number of nodes in a cluster is 4096/512=8 on average. Figure 13(c) shows the logical communication cost of each scheme. Comparing Figure 13(c) to Figure 13(a), we find that the costs of MAAN, SWORD and Mercury keep nearly the same. This is because their costs are mainly affected by the long path lengths in system-wise routing. We notice that the cost of HCO/o increases from around 3000 when d= 11 to around 9000 when d= 9, while that of HCO increases from around 4200 to 5000.

To analyze the reason, we drew Figure 13(d).

The figure shows that more percentages of requests travel longer distance in HCO/o and HCO. Specifically, 93% of messages travel 3 hops and 100% of messages travel 4 hops in HCO/o, 27% of messages travel 7 hops and 100% of messages travel 9 hops in HCO. This is because when the number of nodes in a cluster is 8 on average, a request travels 4 hops before arriving at a destination in the cluster, and travels 8 hops for a cluster circulation on average. Also, we see there is no request that is forwarded within 4 hops in HCO. This implies that the number of nodes in a cluster is at least 5 hops. In HCO/o, each request needs to travel $0-4$ hops. HCO/o combines the requests in one cluster into one message and each message travels $5-9$ hops. Thus, HCO reduces the logical communication cost by reducing the number of messages in routing. In a cluster with three nodes, a message needs to circulate 3 hops in HCO, and the total number of hops traveled by all requests in HCO/o is also 3. When the number of nodes in a cluster is larger than 3, the number of traveling hops in HCO is less than that in HCO/o. This demonstrates that the cluster-token forwarding algorithm is effective in reducing cost with a high node density (>3) in a cluster.

Figure 14(a) plots the resource request success rates against node arrival/departure rate. Because the performance of HCO and HCO/o are similar, we use HCO to represent both. The success rate decreases monotonically in all resource management schemes. MAAN and SWORD incur lower success rates than HCO and Mercury. Because of node and resource dynamism, some requests may not be forwarded to the directory node having the requested resources. More requests fail to arrive at their destinations when the node arrival/departure rate increases, leading to a decrease in the success rate. HCO and Mercury distribute resource descriptors among all nodes in the system, while MAAN and SWORD mainly depend on 11 nodes as shown in Figure 11(c). Relying on a small number of directory nodes increases the possibility of a request failure because only one node departure in the directory nodes will result in the loss of a large number of resource descriptors, resulting in sharp drop-off of the success rate. By amortizing the failure risk among a large number of directory nodes, HCO and Mercury lead to higher success rates.

The efficiency performance in dynamism is also reflected in the number of probed nodes for resource requests. When a request arrives at its destination, if the destination does not have the descriptors of the requested resource, then its nearby nodes are probed for the requested resource. This is because the destination may be a newly joined node that has not received its responsible descriptors from other nodes, or it may be a departing node that has transferred its descriptors to its nearby nodes but other nodes have not updated their leaf sets for its departure. Also, in HCO, other clusters need to be probed when a request cannot be satisfied in the requester's cluster. Figure 14(d) shows the number of probed nodes for requests of each resource management scheme. Because the performance of HCO and HCO/o are similar, we use HCO to represent both. We can see that the numbers are almost constant in different dynamism degrees since node leaf sets are always updated. SWORD and Mercury probe fewer nodes than HCO. This is expected because HCO has an extra probing phase to probe nodes for resources. MAAN has to probe more nodes than others due to its two lookups per query. These experimental results verify the superior performance of Mercury and HCO, compared with MAAN and SWORD in handling network dynamism.



(a) Overdue responses without churn (b) Overdue responses with churn

Fig. 15. Effectiveness of the deadline-driven algorithms.

5.6 Deadline-driven Resource Management

This experiment aims to show the effectiveness of deadline-driven resource allocation and message forwarding algorithms in HCO. We call the HCO scheme with the deadline-driven algorithms *Deadline-driven HCO*. In the experiment, the available time of an available resource and the deadline of a node's resource request were randomly generated in the range of [0,1000] time units, and we set the deadline threshold for urgent requests to 10 time units. Different parameter settings will not change the relative performance differences between HCO and *Deadline-driven HCO*. An overdue response occurs when a resource requester receives a response of an allocated resource after the deadline.

Figure 15(a) illustrates the number of overdue request responses in HCO and *Deadline-driven HCO* versus the number of requests. The results show that HCO has many more overdue responses than *Deadline-driven HCO*, which demonstrates the effectiveness of the deadline-driven resource allocation and message forwarding algorithms. In *Deadline-driven HCO*, a directory node processes urgent requests immediately without waiting for the next periodical resource allocation. Also, a directory node first handles requests with earlier deadlines when mapping the requested resources in RRL to the available resources in ARL. In addition, message forwarders in routing for requests and notifications give higher priorities to requests with earlier deadlines. As a result, the requests with earlier deadlines can be resolved earlier. Without the deadline-driven algorithms, HCO is not able to process some requests with early deadlines in time, leading to more overdue request responses.

Dynamism in networks poses a challenge to resource management schemes to process resource requests in time. This is because frequent node joins and departures lead to a delay in message routing. We assume that one hop forwarding in routing generates a latency of 1 time unit. Figure 15(b) plots the number of overdue request responses in HCO and *Deadline-driven HCO* versus the node interarrival/interdeparture rate. The figure shows that the number of overdue request responses increases as the node interarrival/interdeparture rate increases. A message needs longer time to arrive at its destination in an environment with higher node dynamism. Consequently, more requests cannot arrive at the directory nodes and more notifications cannot arrive at the resource requesters in time. We can also see that *Deadline-driven HCO* reduces the overdue request responses in HCO. This means that the deadline-driven algorithms are effective in decreasing the number of overdue request responses in node dynamism.

6 Conclusions

Rapid development of Grids demands a scalable and efficient resource management scheme to sustain distributed performance in a dynamic wide-area environment. The major contributions of this work are summarized below: (a) This paper

presents a HCO by extending the cycloid DHT overlay. The HCO pools physically close resources together in logically-close clusters. (b) We have developed locality-preserving algorithms to enable users to dynamically discover physically close resources with required functionalities in their neighborhood. Most previous schemes fail to preserve the locality and require users to discover resources on a system-wide scope. (c) The HCO scheme uses a single large DHT overlay with low overhead. It achieves a balanced workload distribution and resilience to resource failure. Most previous schemes use multiple DHT-based overlays causing high overhead or one DHT overlay causing a workload imbalance. Both of those are more suitable for static Grid configurations with limited applications. (d) The cluster-token forwarding and deadline-driven resource management algorithms enhance system efficiency. Both analytical and simulation results demonstrate the superior performance of HCO in Grid reconfiguration for large-scale and dynamic applications.

The proposed framework is still under intensive system and middleware development. For further research, we will continue our efforts in the following aspects: (1) Prototyping of the proposed HCO network for Grid resource management. (2) Developing benchmark programs to test the efficiency and validate the claimed advantages. (3) Applying virtual machine techniques [4] to extend the HCO model to secure and safeguard Grid applications. (4) Integrating P2P and Grid technologies with machine virtualization techniques for global-scale Internet applications. These four areas post wide open problems that are crucial to promote large-scale distributed computing in the future.

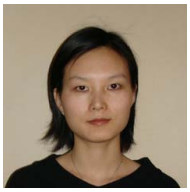
Acknowledgements

This research was supported in part by U.S. NSF grants OCI-1064230, CNS-1049947, CNS-1025652, CNS-1025649, and CNS-0917056, Microsoft Research Faculty Fellowship 8300751, Sandia National Laboratories grant 10002282, and China's 973 Basic Research Grant 2011CB302505. Kai Hwang wants to thank the support of his academic visits of Tsinghua University by Intellectual Ventures, Inc. since 2010. An early version of this work [55] was presented in the Proceedings of ICDCS'09.

References

[1] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: a computation management agent for multiinstitutional grids. In *Proc. HPDC*, 2001.
[2] I. Foster and C. Kesselman. Globus: a metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2), 1997.
[3] F. Berman et. al. Adaptive computing on the grid using apples. *TPDS*, 14(4), Apr. 2003.
[4] J. Smith and R. Nair. Virtual Machines,Virtual Machines: Versatile Platforms for Systems and Processes. *Morgan Kaufmann Publisher*, 2005.
[5] I. Stoica, R. Morris, and et al. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM TON*, 11(1):17–32, 2003.
[6] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*, 2001.
[7] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: An Infrastructure for Fault-tolerant wide-area location and routing. *J-SAC*, 12(1):41–53, 2004.
[8] H. Shen, C. Xu, and G. Chen. Cycloid: A Scalable Constant-Degree P2P Overlay Network. *Performance Evaluation*, 63(3):195–216, 2006.
[9] D. Karger and et al. 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.
[10] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proc. of ACM SIGCOMM*, pages 353–366, 2004.
[11] M. Cai, M. Frank, and et al. MAAN: A multi-attribute addressable network for grid information services. *Journal of Grid Computing*, 2004.
[12] D. Oppenheimer, J. Albrecht, and et al. Scalable wide-area resource discovery. Technical Report TR CSD04-1334, EECS Department, Univ. of California, Berkeley, 2004.
[13] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *Proc. of HPDC*, pages 226–235, 2003.
[14] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Proc. of P2P*, 2002.
[15] D. Spence and T. Harris. Xenosearch: Distributed resource discovery in the XenoServer open platform. In *Proc. of HPDC*, 2003.
[16] Z. Zhang, S.-M. Shi, and J. Zhu. Somo: Self-organized metadata overlay for resource management in P2P DHT. In *Proc. of IPTPS*, 2003.
[17] H. Shen. PIRD: P2P-based Intelligent Resource Discovery in Internet-based Distributed Systems Corresponding. *JPDC*, 2008.
[18] M. Cai and K. Hwang. Distributed aggregation algorithms with load-balancing for scalable grid resource monitoring. In *Proc. of IPDPS*, 2007.
[19] S. Suri, C. Töth, and Y. Zhou. Uncoordinated load balancing and congestion games in P2P systems. In *Proc. of P2P*, 2004.
[20] H. Shen and C. Xu. Locality-aware and churn-resilient load balancing algorithms in structured peer-to-peer networks. *TPDS*, 2007.
[21] Y. Zhu and Y. Hu. Ferry: A P2P-Based Architecture for Content-Based Publish/Subscribe Services. *TPDS*, pages 672–685, 2007.
[22] C. Schmidt and M. Parashar. Squid: Enabling Search in DHT-based Systems. *JPDC*, pages 962–975, 2008.
[23] Y. Zhu, H. Jiang, J. Wang, and F. Xian. HBA: Distributed Metadata Management for Large Cluster-Based Storage Systems. *TPDS*, 2008.
[24] J. Wang, P. Gu, and H. Cai. An Advertisement-based Peer-to-Peer Search Algorithm. *JPDC*, pages 638–651, 2009.
[25] C. Wang and X. Li. An Effective P2P Search Scheme to Exploit File Sharing Heterogeneity. *TPDS*, pages 145–157, 2007.
[26] R. Zhang and Y. C. Hu. Assisted Peer-to-Peer Search with Partial Indexing. *TPDS*, pages 1146–1158, 2007.
[27] M. Li, W.-C. Lee, A. Sivasubramaniam, and . Zhao. SSW: A Small-World-Based Overlay for Peer-to-Peer Search. *TPDS*, 2008.
[28] X. Shi, J. Han, Y. Liu, and L. Ni. Popularity Adaptive Search in Hybrid P2P Systems. *JPDC*, pages 125–134, 2009.
[29] E. Pagani, G. Rossi, and E. Pertoso. ORION - Ontology-based queRy routIng in Overlay Networks. *JPDC*, pages 28–38, 2009.
[30] F. P. Preparata and J. Vuillemin. The cube-connected cycles: A versatile network for parallel computation. *CACM*, 24(5):300–309, 1981.
[31] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. of INFOCOM*, 2002.
[32] Z. Xu, M. Mahalingam, and M. Karlsson. Turning heterogeneity into an advantage in overlay routing. In *Proc. of INFOCOM*, 2003.
[33] 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.
[34] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *Proc. of SIGMOD*, 1990.
[35] P. Ganesan, B. Yang, and H. Garcia-molina. One torus to rule them all: Multi-dimensional queries in p2p systems. In *Proc. of WebDB*. ACM Press, 2004.
[36] S. Chenga and T.-L. Wua. Fast indexing method for image retrieval using k nearest neighbors searches by principal axis analysis. *Journal of Visual Communication and Image Representation*, 2006.
[37] L. He, L. Wu, Y. Cai, and Y. Liu. Indexing Structures for Content-Based Retrieval of Large Image Databases: A Review. *Lecture Notes in Computer Science*, 2005.
[38] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. *Future Directions in Distributed Computing*, 2002.
[39] P. Salvador and A. Nogueira. Study on geographical distribution and availability of bittorrent peers sharing video files. In *Proc. of ISCE*, 2008.
[40] K. Czajkowski and et al. Resource Specification Language (RSL). *Globus*, 1998.
[41] P. Ganesan and K. Respectively. Canon in G major: Designing DHTs with hierarchical structure. In *Proc. of ICDCS*, pages 263–272, 2004.
[42] S. Zoels, Z. Despotovic, and W. Kellerer. Cost-based analysis of hierarchical DHT design. In *Proc. of P2P*, pages 6–8, 2006.
[43] C. Chen and D.P. Agrawal. dBCube: a new class of hierarchical multiprocessor interconnection networks with area efficient layout. *TPDS*, 4(12):1332 – 1344, 1993.
[44] D. Guo, J. Wu, H. Chen, and X. Luo. Moore: An extendable peer-to-peer network based on incomplete kautz digraph with constant degree. In *Proc. of the IEEE International Conference on Computer Communications (INFOCOM'07)*, page 821.
[45] D. Guo, H. Chen, Y. Hec, H. Jin, C. Chen, H. Chen, Z. Shu, and G. Huang. Kcube: A novel architecture for interconnection networks. *Information Processing Letters*, 110(18-19):821–825, 2010.
[46] D. Guo, H. Chen, Y. Liu, and X. Li. Bake: A balanced kautz tree structure for peer-to-peer networks. In *Proc. of INFOCOM*, 2008.
[47] Q. Li, L. Feng, J. Pei, S.X. Wang, X. Zhou, and Q. Zhu. Advances in Data and Web ManagementJoint International Conferences. *Lecture Notes in Computer Science*, 2009.
[48] H. Shen and C. Xu. Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured Peer-to-Peer Networks. *TPDS*, 2007.
[49] K. Psounisa, P. M. Fernandezb, B. Prabhakarc, and F. Papadopoulosd. Systems with multiple servers under heavy-tailed workloads. *Performance Evaluation*, 2005.
[50] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Trans. Network*, 1997.
[51] B. Krishnamurthy and J. Rexford. Web Protocols and Practice (Chapter

10). Addison Wesley, 2001.
[52] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. *TOCS*, 1997.
[53] X. Zhang, Y. Qu, and L. Xiao. Improving distributed workload performance by sharing both CPU and memory resources. In *Proc. of ICDCS*, pages 233–241, 2000.
[54] E. Zegura, K. Calvert, and et al. How to model an Internetwork. In *Proc. of INFOCOM*, 1996.
[55] H. Shen and K. Hwang. Scalable Grid Resource Discovery with Locality-Preserving Clustering. In *Proc. of the 27th International Conference on Distributed Computing Systems (ICDCS)*, 2009.



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

Kai Hwang Kai Hwang received the Ph.D. in Electrical Engineering and Computer Science from the University of California, Berkeley in 1992. He is presently a Professor of Electrical Engineering and Computer Science, University of Southern California, Los Angeles. He directs the Internet and Cloud Computing Research Laboratory at USC. Prior to joining USC, he has taught at Purdue University for a decade. He has served as a visiting chair professors at the University of Minnesota, University of Hong Kong and National Taiwan University. Presently, he also serves as an endowed Visiting Chair Professor of Tsinghua University, Beijing, China. He has served as the editor in Chief of the Journal of parallel and Distributed Computing and Associate Editor of the IEEE Transactions on Parallel and Distributed Systems. An IEEE life Fellow, Hwang specializes in computer systems, parallel processing, Internet security, distributed and cloud computing. He has published 8 books and over 220 scientific papers in these areas. Contact him at: kaihwang@usc.edu.