A DHT-Based Grid Resource Indexing and Discovery Scheme

Yong Meng TEO[1,2], Verdi March[2] and Xianbing Wang[1]

[1]Singapore-MIT Alliance, [2]Department of Computer Science, National University of Singapore

***Abstract*— This paper presents a DHT-based grid resource indexing and discovery (DGRID) approach. With DGRID, resource-information data is stored on its own administrative domain and each domain, represented by an index server, is virtualized to several nodes (virtual servers) subjected to the number of resource types it has. Then, all nodes are arranged as a structured overlay network or distributed hash table (DHT). Comparing to existing grid resource indexing and discovery schemes, the benefits of DGRID include improve security of domains, increase availability of data, and elimination of stale data.**

***Index Terms*— Grid, resource indexing and discovery, DHT, availability**

I. Introduction

Grid computing facilitates resource sharing and collaboration across multiple administrative domains. An *administrative domain* is a collection of grid resources maintained by a single administrative authority. A computational grid, which consists of compute resources distributed across administrative domains, allows the execution of compute-intensive applications across domain. Each domain deploys a publicly-accessible index server such as Globus MDS [2], which registers resources and maintains resource information such as processor, memory, operating system, number of compute nodes, etc.

Resource discovery is an important step in grid resource management [6], [12], [18]. A grid scheduler determines the set of resources to match the application execution requirements. This requires an information system that supports the efficient indexing and discovery of grid resources. The information system must be scalable in the presence of many administrative domains.

MDS [2], an OGSI [3] implementation of information services, uses the centralized architecture to aggregate information from various sources. The scalability of centralized architecture limits MDS to a small number of administrative domains in a small grid.

A highly scalable information system should be decentralized, where index servers from various administrative domains cooperate to form a distributed information system. Index servers can be organized as an unstructured overlay network or structured overlay network. In unstructured networks, it is hard to theoretically bound the average lookup performance, in term of number of hops in the overlay network. Structured overlay network or distributed hash table (DHT) aims to provide efficient and scalable lookup performance in a distributed system. It also provides a stronger result guarantee than unstructured overlay networks. However, the lookup performance and stronger result guarantee can be achieved only if the structured overlay network is well-maintained.

DHT works by organizing nodes as a topology and distributing data to designated nodes – the owner of data cannot interfere with the placement of data. The grid information system can be implemented using DHTs such as [13], [14], [17], where data is an index to a resource type in an administrative domain and a node is an index server that stores data of the same type regardless the origin of the data. Not only this raises the security issue, but also (i) to increase the availability of data, a node needs to replicate all its data to other nodes, otherwise when the node fails, all resources of a certain type cannot be discovered, (ii) data may point to a domain disconnected from the grid due to network failure, in which resources in the disconnected domain cannot be utilized anyway, and (iii) users need to separately query the index server in the resource's domain for detailed resource information.

SkipNet [8] enforces *content locality* to control the data placement to address the security and proper-usage issues. However, users must know the data domain when searching as SkipNet prefixes data identifier with a domain name. In a computational grid, users should be allowed to search multiple resources of the same type without enumerating domain names.

DHT-based Grid Resource Indexing and Discovery (*DGRID*) is a DHT-based framework for indexing grid resources. The main features of DGRID are (i) data resides on the domain that owns it instead of redistributed in the information system infrastructure, (ii) supports lookups for resources that span multiple administrative domains without specifying domain names, and (iii) reuse existing data-indexing and DHTs algorithms.

DGRID makes two assumptions: (i) the total number of compute-resource types in a computational grid is small, i.e. several thousands or less, and there is a significant overlap of resource types across domains; (ii) the total number of

Yong Meng TEO is with the Department of Computer Science, School of Computing, 3 Science Drive 2, National University of Singapore, Singapore 117543, and Singapore-MIT Alliance, 4 Engineering Drive 3, National University of Singapore, Singapore 117576 (e-mail: teoym@comp.nus.edu.sg).

Verdi March is with the Department of Computer Science, School of Computing, 3 Science Drive 2, National University of Singapore, Singapore 117543 (email: verdimar@comp.nus.edu.sg).

Xiangbing Wang is with Singapore-MIT Alliance, 4 Engineering Drive 3, National University of Singapore, Singapore 117576 (e-mail: wangxb@comp.nus.edu.sg).

compute-resource *types* per administrative domain is smaller than the number of possible compute-resource types.

DGRID virtualizes an index server into a number of nodes (virtual servers) subjected to the number of resource types registered on it. Then, nodes are arranged as a DHT.

The contributions of this paper are the design of DGRID framework, a DGRID implementation using Chord, theoretical analysis for the overhead and lookup performance of Chord-based DGRID, and an evaluation of Chord-based DGRID via simulations.

The rest of this paper is organized as follows. Section II describes the design of DGRID and its Chord-based implementation, section III presents our theoretical analysis, section IV presents an experimental evaluation and discussion. Section V describes the related works, and section VI concludes this paper.

II. DGRID

Let a computational grid $G = \{d\}$ where d represents an administrative domain. Each administrative domain is defined as $d = (S, R, T)$ where S is an index server such as MDS [2], R is a set of compute resources, and T is a set of *resource types*, $T = \{t\}$ and $|T| \leq |R|$. A resource is characterized by the resource type $t = \{a\}$ where $a(attr\_type, attr\_value)$ represents a static attribute, such as `(CPU Speed, 1 GHz)` or `(Memory, 1 GB)`.

Index server S contains indices to all resource types in the administrative domain, $S = \{r\}$. An index r is defined as $r = (t, d)$, which denotes that r is an index to resource type t in administrative domain d. There is a one-to-one correspondence between S and T, which means (i) $|S||T|$, (ii) each index $r \in S$ corresponds to exactly one resource type $t \in T$, and (iii) each $t \in T$ is represented by exactly one index $r \in S$. Since a lookup targets and fetches indices, indices are also referred as *data* where the key is the resource type t. Each data is assigned an identifier based on its key such that $id(r) = id(t)$. Methods to produce data identifier given a key are given in [4], [15]. In a computational grid, there can be many domains providing resources of type t, and hence, many index servers storing indices identified by $id(t)$.

A. The DGRID Scheme

DGRID is a DHT-based node-organizing framework that supports two additional constraints:

1) Data is stored at its originated location, i.e. $r = (t, d)$ must be stored on S_d, the index server S of domain d.
2) Allow users to perform lookup without specifying domain name.

Given a domain $d = (S, R, T)$, DGRID virtualizes a index server S into $|T|$ nodes (virtual servers); each node represents one index $r \in S$. The virtualization of index servers ensures that data are not redistributed to other domains. However, we need to ensure that virtualization avoid nodes collision (figure 1).

The index server of a domain is responsible for organizing resource information within the domain, similar as MDS [2]. In DGRID, each index server only need to provide the types and the number of resources of each type to others.

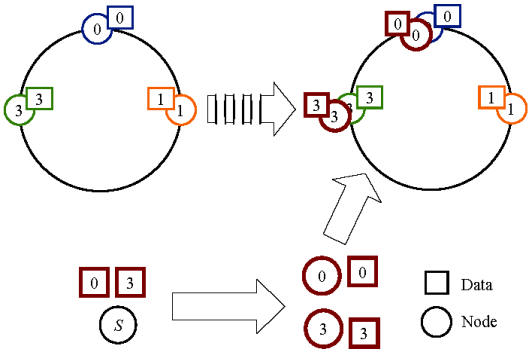

Fig. 1. Virtualizing Index Servers – Collisions must be Avoided

B. Identifier Definition

To avoid nodes collision, we split the node identifier into two parts: a prefix denotes a data identifier and a suffix denotes an index-server identifier (figure 2). Given node n representing $r = (t, d)$, the m-bit identifier of n is the combination of i-bit identifier of t, where $i \leq m$, and $m - i$-bit identifier of S, which is expressed as $id_m(n) = id_i(t) \oplus id_{m-i}(S)$. DGRID guarantees that all $id_m(n)$ are unique, give any two nodes, their identifiers differ either in their prefixes or suffixes.

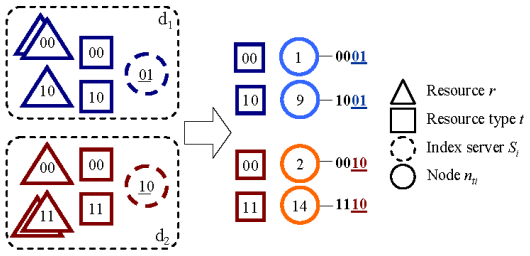

Fig. 2. Node Identifier Comprised of Identifiers of Data and Index Server

Since DGRID controls the generation of $id_m(n)$, the underlying DHT must not dynamically modify $id_m(n)$. Therefore, DHT such as CAN [13] cannot be used directly as the underlying DHT for DGRID framework because CAN dynamically modifies the node identifier. Modifications on CAN are required to implement DGRID, but this is beyond the scope of this paper.

C. Setup of DGRID

`setup` is the process required by an index server S to become a part of a grid information system. In DGRID, `setup` is the virtualization of an index server S into $|T|$ virtual servers. Each virtual server joins a DGRID system to become a node in the DGRID system; this process is referred as `join`. Thus, a `setup` in DGRID consists of $|T|$ `join`. Figure 3 shows how S joins a DGRID system via an existing node e.

The `join` operation ensures that the node identifiers do not collide, by combining $id_i(t)$ and $id_{m-i}(S)$ as the node identifier. Then, the node joins the DGRID using the join protocol of the underlying DHT, `joinDHT`. Figure 4 shows the `join` operation, which delegates the rest of the join

```
setup(IndexServer S, Node e)
  for each r ∈ S do
    Let n := virtual server of S that stores r
    join(n, r, e)    //Perform collision avoidance
```

Fig. 3. DGRID Setup Operation

process to the underlying DHT's join protocol, which is Chord.

```
join(Node n, ResourceType t, Node e)
  Let S := index server associated with n
  Let id(n) := id(t) ⊕ id(S)
  n.joinDHT(e)    //Chord's join operation
```

Fig. 4. DGRID Join Operation

D. Lookup Services

To facilitate searching, DGRID provides `get(key)` interface similar to the one provides by other DHTs. DGRID will route each query to any nodes that hold the answer. Additional constraints related to more advance query processing, i.e. answers must be provided by the least-loaded nodes or physically-nearby nodes, can be implemented on top of DGRID. Although DGRID allows users to search resources without specifying domains, DGRID must also support users who need n resources of type t, but the resources must be provided by domains specified in $D_u = d$. To efficiently support such lookup requests, we modify DGRID so that an index server S of domain d is identified by $id'_{m-i}(S) = id_j(d) \oplus id_{m-i-j}(S)$, $j < (m-i)$. Then, q is routed to a node n that maps to S where $prefix_j(id'_{m-i}(s)) = id_j(d)$, $d \in D_u$.

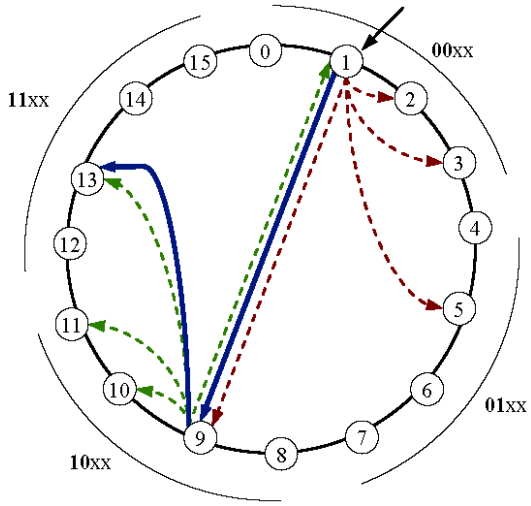

Fig. 5. Lookup in Chord-based DGRID

Figure 5 illustrate the lookup operation of Chord-based DGRID. To lookup data, a query q for a resource type identified by $id_i(t)$ is translated to q', a query for data identified with $id_m(q') = id_i(t) \oplus 0$. This is because $id_i(t)$ is i-bit length, whereas the identifier space is m-bit length. Therefore, we need to pad $id_i(t)$ with $(m-i)$-bit digits to produce an m-bit identifier. With this scheme, the lookup cost is bounded by the lookup cost of the underlying DHT (e.g. $O(log\ N)$ in Chord). Several optimizations can be performed for the lookup:

1) Not only the node identified with $id_i(t) \oplus 0$ provides the answer for q', but also any node n whose $prefix_i(id_m(n)) = id_i(t)$.
2) If node n is associated to a index server S and $prefix_i(n) \neq id_i(t)$, then n can check if S owns $r = (t, d)$ where $id_i(r) = id_i(t)$. If so, then q' is completed at n since n contains the answer for q'.
3) When deciding the node where q' will be forwarded, node n checks if its routing table contains n' whose $prefix(id(n')) = id(t)$. If n' exists, then forwards q' to n', otherwise, forward q' to the node chosen with the routing protocol of the underlying DHT.
4) Node n can checks all the routing tables (each routing table correspond to one node) maintained by its associated index server S. This increases the flexibility of path selection.

Figure 6 shows the lookup algorithm of Chord-based DGRID. In this algorithm, node n is asked to perform a lookup for data of type t.

```
get(Node n, ResourceType t)
  if (a := containsData(n, t)) ≥ 0 then
    return a
  else if (f := fingerContainsData(n, t)) ≥ 0 then
    get(f, t)
  else
    Let succ := n.find_successor(t ⊕ id_{m-i}(0))
    get(succ, t)
containsData(Node n, ResourceType t)
  Let S := index server identified by suffix(id(n))
  Let A := list of data stored at S
  for i := 1 to |A| do
    if id(A[i]) = id(t) then
      return A[i]

  return -1
fingerContainsData(Node n, ResourceType t)
  Let F := finger table of n
  for i := 1 to |F| do
    Let fid := id(F[i])
    if prefix(fid) = id(t) then
      return fid;

  return -1;
```

Fig. 6. DGRID Lookup Operation

In DGRID, nodes with the same prefix are located together in a segment. This eases the processing of lookups whose answers come from many index servers. Suppose there is a query q to find $|R_q|$ resources of type t identified by $id_i(t)$. DGRID converts q to q', a query for resource type identified by $id_m(t) = id_i(t) \oplus 0$. Eventually, q' will arrive at n whose $prefix_i(id_m(n)) = id_i(t)$ and the index server S associated with n has $|R_n|$ resources, $|R_n| < |R_q|$. To find the rest $|R_q| - |R_n|$ resources, n can simply propagate q to nodes in segment $id_i(n)$, which means all nodes n' whose $prefix_i(id_m(n')) = prefix_i(id_m(n)) = id_i(t)$. It can be

guaranteed that all nodes within the same segment corresponds to different index servers.

III. Analysis

In this section, we analyze the performance of a Chord-based DGRID, subsequently referred as C-DGRID, and compare it to a scheme that uses Chord to manage the resource index and discovery, subsequently referred simply as Chord. One main issue with Chord is unbalanced data distribution when the number of total resource types is small. Table I shows the comparison between Chord and C-DGRID. Let the total number of nodes as $N = 2^m$, the total number of resource types as $Y = 2^i$ where $i < m$, the total number of nodes per segment as $M = \frac{N}{Y}$, and the average number resource types per domain as $0 \leq g \leq Y$, i.e. given domain $d(S, T, R)$, $|T| = g$.

A. Setup Cost

In Chord, the setup cost equals to $O(log^2\ N+g{\cdot}log\ N+X)$, which equals to one DHT join operation, g stores, and possibly redistributions of copies of data to the joining node. The costs of one join and one store are $O(log^2\ N)$ and $O(log\ N)$, respectively, and possibly distribute some data to the joining node and X denotes the cost.

In C-DGRID, the setup cost equals to $O(g{\cdot}log^2\ N)$ because the virtualization of one domain requires g node joins and the cost of one join is $O(log^2\ N)$.

B. Add a New Resource Type

In Chord, adding a new resource type is equivalent to a store operation, which is $O(log\ N)$. In C-DGRID, adding a new resource to a domain that already has the resource type will only refresh the resource information in the index server. However, adding a new resource type causes a new node to join the C-DGRID system, which is $O(log^2\ N)$. However, the number of resources with the same type may change more frequently than adding a new resource type within a domain. So, the store operation of Chord would be invoked more frequently than the join operation in C-DGRID.

C. Total States per Indexing Server

In Chord, each indexing server maintains $O(log\ N)$ states (fingers). In C-DGRID, an index server maintains $O(g{\cdot}log\ N)$ because it is virtualized to g nodes and each node maintains $O(log\ N)$ states.

D. Finger Flexibility

Although each domain in C-DGRID maintains more states than in Chord, it is not necessary for nodes in C-DGRID to maintain the same level of correctness as nodes in Chord. The reason is as follows. In Chord, the successor of n is $n' \geq n$ and no n'' exists such that $n \leq n'' < n'$. The i^{th} finger of node n is the successor of $n + 2^{i-1}$, which amounts to $O(1)$ node. In C-DGRID, the successor of n can be any node whose shares the same prefix, $prefix(id(n))$. The i^{th} finger of node n can be any node within the segment $prefix(n + 2^{i-1})$, which amounts to $O(M)$ nodes.

Figure 7 shows the finger flexibility in Chord and C-DGRID. The dashed edges denote the i^{th} finger of x according to Chord definition, which is n. In C-DGRID, in addition to n, x is allowed to point to n' or n'' as its i^{th} finger (solid edges).

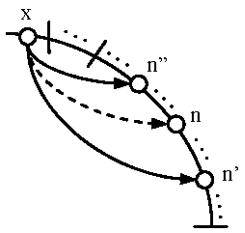

Fig. 7. Finger Flexibility in C-DGRID

E. Incoming Edges to Data

There are more incoming edges to data in C-DGRID than in Chord. In Chord, data is available in a particular node and the node has $O(log\ N)$ incoming edges. In Chord-based DGRID, data is available in a segment. Since a segment consists of $O(M)$ nodes, a segment has $O(M \cdot log\ N)$ incoming edges. However, some of the edges come from nodes within the segment itself. More incoming edges increases the reachability of the segment and reduces the path length taken by lookup requests.

F. Availability

C-DGRID increases the availability of data as each data type is available at $O(M)$ nodes of a particular segment. As each node holds only one copy per data, the number of data stored is $O(M)$. In Chord, we need to replicate all data to several neighbors. The benefits of our design are:

1) Increase fault tolerance.
   If a node of a segment fails, there are still $O(M-1)$ in the segment; all of which provide the same data as the one provided by the failed node.
2) No overhead of replication, because only one copy of each resource information exists in C-DGRID.
3) Eliminates stale data.
   In Chord, data is stored not at its originated domain. Hence, if the originated node fails, clients may get stale data. In C-DGRID, data is stored at its originated domain. If the domain fails, clients will never be given data from the failed domain. This is useful in computational grid because clients cannot utilize resources in the failed domain as pointed by the stale data.

G. Lookup Cost

In computational grid, a client may issue a lookup for resource type t and expect C copies of answers – each copy describes one domain that provides resources of type t. The lookup cost essentially captures the cost to reach the nodes that store the data (number of hops).

TABLE I
COSTS COMPARISON

Operation	Chord	Chord-based DGRID
Join	$O(log^2\ N)$	$O(log^2\ N)$
Setup	$O(log^2\ N + g \cdot log\ N + X)$	$O(g \cdot log^2\ N)$
Add new resource type	$O(log\ N)$	$O(log^2\ N)$
Total state per server	$O(log\ N)$	$O(g \cdot log\ N)$
Finger flexibility	$O(1)$	$O(M)$
Find one	$O(log\ N)$	$O(log\ Y)$
Find all	$O(log\ N)$	$O(log\ Y + M)$

In Chord, the cost to reach the node that stores the data is $O(log\ N)$. No additional hops are needed because all copies of the data are stored on one node. Hence, for any value of C, the lookup cost is always $O(log\ N)$.

In C-DGRID, the cost to reach the segment representing the data is $O(log\ Y)$. Since a node may not hold C copies, the lookup request needs to be propagated to $C-1$ nodes in the segment, which requires $O(C)$ hops. Hence, the lookup cost becomes $O(log\ Y + C)$. If the client wants to retrieve only one copy, the lookup cost becomes $O(log\ Y)$, otherwise if the client wants to retrieve all copies, then the lookup cost becomes $O(log\ Y + M)$.

The cost of reaching a segment is only $O(log\ Y)$ is because every node maintains fingers to $O(log\ Y)$ segments. Therefore, C-DGRID routes a lookup request from one segment to another segment, and each time halves the distance (in term of *segment*) to the destination. The formal proof then follows the one given in [17].

The reason node n maintains fingers to $O(log\ Y)$ segments is as follows. Of $O(log\ N)$ n's finger, up to $log\ M$ points to the same segment S. The rest of the fingers will point to different segments, e.g. the $(log\ M + x)^{th}$ finger points to a node in segment $S + 2^{x-1}$. Obviously, the distance between S and the segment pointed by the $(log\ M + x)^{th}$ finger is twice the distance between S and the segment pointed by $(log\ M + x - 1)^{th}$ finger.

IV. SIMULATION

To further study the performance of DGRID, we modified the Chord simulator [1] to simulate C-DGRID. We compare the overhead, lookup performance, and resilience to failures of Chord and C-DGRID. In both Chord and C-DGRID simulators, we disable the caching of fingers and modify how lookups are processed (see section IV-B for more detail). We simulate a grid of 50,000 domains; each domain has on average g distinct resource types and is represented by one index server. We use $m = 24$-bit and $i = 8$-bit ($Y = 256$). The request forwarding between physical index servers is penalized 50 ms (exponentially distributed) and the request processing by each node is penalized by [5, 15] ms (uniformly distributed).

A. Overhead

To evaluate the overhead of Chord and C-DGRID, we measure the average bootstrap time and the convergence time.

The *bootstrap time* of node n is defined as $b_n = time_{recognized} - time_{arrive}$, where $time_{arrive}$ = time when n arrives and $time_{recognized}$ = time when n is recognized by another node n', i.e. n becomes the predecessor of n'.

The stabilization degree of the whole system is defined as

$$S = \frac{\sum_{n=0}^{N-1} s_n}{N} \qquad 0 \le S \le 1$$

where N is the total number of nodes. The *stabilization degree* of node n is defined as

$$s_n = \begin{cases} 0 & \text{if } n \text{ has an incorrect } 1_{st} \text{ successor} \\ \frac{|F'|+|U'|}{m+max_U} & \end{cases}$$

where $0 \le s_n \le 1$, F' is the proper fingers in finger table, max_F is the length of node identifier ($0 \le |F'| \le max_F$), U' is the proper successors in successor list, and max_U is a constant ($0 \le |U'| \le max_U$). The i^{th} proper finger and successor of n is the immediate successor of $(n + 2^{i-1})$ and $(n+i)$, respectively. Suppose that at time $time_0$, the i^{th} proper finger and successor of n is n_a and n_b, respectively. If n_c joins at time $time_1 > time_0$ and $(n + 2^{i-1}) \le n_c < n_a$, then n_c will become the i^{th} proper finger of n. Similarly, if n_d joins at time $time_1 > time_0$ and $(n + i) \le n_d < n_b$, then n_d will become the i^{th} proper successor of n. If n updates its proper finger and successor to n_b and n_d at time $time_2 > time_1$, then $s_n < 1$ during the interval $time_2 - time_1$.

The *convergence time* of the system is defined as $c = time_S - time_{last\_arrival}$ where $time_{last\_arrival}$ is the time when the last node arrives and $time_S$ is the time when the system reaches the desired S.

The simulations to compare the stabilization convergence in Chord and C-DGRID are performed as follows. Index servers enter a Chord or C-DGRID system according to a Poisson process with 1 second mean arrival rate. Each index servers has U[2, 5] number of resource types, which results in approximately 175,000 nodes in C-DGRID. Each node joins through a randomly chosen existing node. Each node maintains one successor pointer ($max_U = 1$). Each nodes invokes the stabilization mechanism for every $[0.5p, 1.5p]$ seconds interval (uniform distribution). To measure the convergence time c, we calculate S every 1 hours.

Table II shows the average bootstrap time in Chord and Chord-DGRID, under various p (secs).

Figure 8 shows the convergence time c of both systems, which implies the overhead of Chord and C-DGRID. Due to the larger number of nodes, C-DGRID requires a larger c than Chord to reach the same S. However, in defining S we do not consider the finger flexibility of C-DGRID, which means the overhead of C-DGRID presented here is the worst-case

TABLE II
AVERAGE BOOTSTRAP TIME b (SECS)

p	Chord	C-DGRID
60	1.5	3.0
240	41.4	65.8
960	671.8	2,033.8

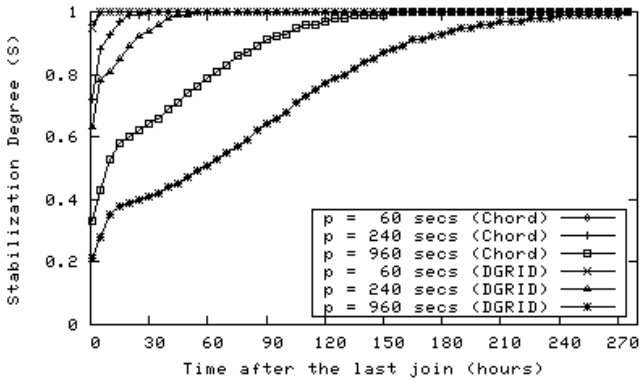

Fig. 8. Convergence time c (Chord), measured every 5 hours from the last join operation

overhead. In addition, with sufficiently small p, C-DGRID can keep its structured overlay network nearly on par with Chord, given the same amount of time after the last join. With $p = 60$ seconds, after the last join, $S_{Chord} = 0.96$ and $S_{C-DGRID} = 0.95$. After one hour, $S_{Chord} = 1$ and $S_{DGRID} = 0.99$. With $p = 960$ seconds, after the last join, $S_{Chord} = 0.33$ and $S_{C-DGRID} = 0.21$. However, while Chord requires 133 hours to reach $S_{Chord} = 0.99$, C-DGRID requires 239 hours to reach $S_{C-DGRID} = 0.99$, almost twice the time required in Chord.

It is obvious that C-DGRID requires more overhead because it virtualizes each index server into multiple nodes to transparently avoiding data redistribution, increasing the availability of data by positioning an index server in multiple segments in the identifier space, and bounding the query processing of each domain subjected to the resource types in the domain.

B. Lookup Performance

To evaluate the lookup performance of both systems, we simulated lookups on different S. The measurements are the average path length (hops), and the number of failed lookups. The average path length includes only requests forwarded between physical index servers.

First, Chord and C-DGRID networks of 50,000 domains are constructed with every node performs the stabilization every $p = 960$ secs[1]. Then, we perform 500,000 simple lookups (1 second Poisson mean arrival rate). A simple lookup refers to a lookup that can be satisfied by one administrative domain. Each lookup looks for a randomly chosen data and is initiated by a randomly chosen node. In Chord, a lookup for resource type t is successful if the lookup request arrives at node x where either x is the immediate successor of t or x represents a domain with resources of type t. In C-DGRID, a lookup for resource type t is successful if the lookup request arrives at node v where either $prefix_i(id_{i+m}(v)) = id_i(t)$ or v represent a virtual server of a domain who owns resources of type t.

In table III, we vary the number of resource types per domain. The number of resource types is generated using a uniform distribution. The results show that in both cases, C-DGRID has shorter average path length (hops) and fewer failed lookups for various S (except when the average resource types per domain $\sim$ U[2, 5] and $S = 0.33$) although the C-DGRID networks have more nodes (about 175,000 nodes) than the Chord network (50,000 nodes) given the same number of index servers. This is because the expected average path length in C-DGRID is $\frac{1}{2}log\ Y = 4$ as there are Y segments and the node identifiers are randomly distributed. As the average number of resource types per domain is doubled from U[2, 5] to U[4, 10], the average path length in C-DGRID decreases to 2.8 (30% smaller than $\frac{1}{2}log\ Y$) because 1) each domain, on average, is virtualized to 7 nodes instead of 3.5 nodes so that it will be visited more during lookups, and 2) the probability that each node can answer a lookup request is $ln\ \frac{256}{256-7} = 0.028$ instead of $ln\ \frac{256}{256-3.5} = 0.014$.

Note that table III merely shows the lower bound of lookup cost in C-DGRID. For lookups where answers are provided by $|D|$ domains, there are $|D|$ additional hops besides the path length to reach the first node in a particular segment. For instance, if a lookup must be satisfied by 3 domains, then the average path length must be added by 3 hops.

C. Resilience to Random Simultaneous Failures and Leaves

To evaluate the resilience of C-DGRID under simultaneous random failures and leaves, we simulated lookups when a fraction of the network fails or leaves simultaneously. We measure the average path length, the number of failed lookups, and the average number of timeouts.

We start with Chord and C-DGRID networks of 50,000 domains (each domain has $\sim$ U[2, 5] resource types, yielding $g = 3.5$) and let the systems reach a predefined S. Then, a fraction of *domains* fails or leaves simultaneously[2] and the stabilization is deactivated, followed by followed by 500,000 simple lookups (1 seconds Poisson mean arrival rate). For Chord, when a domain fails, it does not have a chance to move data stored to its successor. However, when a node leaves, it instantly moves all data stored to its successor, which is an optimistic assumption. For C-DGRID, data redistribution is not required when a domain fails or leaves.

For a lookup in Chord to be considered successful, in addition to the criteria described earlier in section IV-B, the node that provide the answer to the lookup request must store at least one non-stale data describing the requested resource type, otherwise the lookup is considered failed. Stale data is an index to a resource type in a domains that has failed or left the system.

Table IV shows the results when vary the degrees of node fail and node leave. The fourth and sixth columns in both tables, labeled as *Failed*, refer to the total number of failed

[1]We choose this number to construct networks with low S

[2]In C-DGRID, one domain fail/leave results in several node fails/leaves.

TABLE III
LOOKUP PERFORMANCE WITH VARIOUS STABILIZATION DEGREE S

S	resource type per domain ~ U[2, 5]				resource type per domain ~ U[4, 10]			
	Chord		C-DGRID		Chord		C-DGRID	
	Hops	Failed	Hops	Failed	Hops	Failed	Hops	Failed
0.33	21.9	185	3.4	212	11.6	155	2.8	19
0.40	16.7	162	4.2	86	7.9	144	2.8	10
0.60	6.9	149	3.4	12	7.1	1,434	2.8	2
0.80	7.3	0	3.4	0	7.1	0	2.8	0
1.00	7.3	0	3.4	0	7.1	0	2.8	0

TABLE IV
LOOKUP PERFORMANCE UNDER RANDOM SIMULTANEOUS FAILS AND LEAVES

%Failed	S	Chord		C-DGRID	
		Hops	Failed Lookups	Hops	Failed Lookups
20	0.33	15.8	173,835	5.4	755
	0.40	15.2	140,094	4.5	231
	0.80	7.9	0	4.6	41
40	0.33	9.8	366,172	7.0	5,197
	0.40	10.2	375,704	6.0	817
	0.80	8.6	383,378	6.1	180

%Leave	S	Chord		C-DGRID	
		Hops	Failed Lookups	Hops	Failed Lookups
20	0.33	19.6	139,778	5.3	386
	0.40	15.1	134,222	4.5	125
	0.80	7.9	0	4.6	3
40	0.33	18.0	345,387	6.7	1,794
	0.40	10.6	330,103	5.9	466
	0.80	8.6	1	6.0	101

lookups (false negatives). In most cases, C-DGRID has a lower average path length and fewer failed lookups, except for some cases where $S = 0.8$.

For simultaneous fails, C-DGRID outperforms Chord in term of the number of failed lookups. This is because the Chord networks only has a small number of designated nodes compared to the total number of nodes. Since data identifier is uniformly distributed within [0, 2^8) and node identifier is uniformly distributed within [0, 2^{24}), on the ideal condition ($S = 1$) there are at most two designated nodes and each designated node is responsible for many types of data. If the ratio between designated nodes and total nodes is very small, then it is sufficient to bring down some designated nodes to cause the majority of lookups to fail. Table V shows DN, the number of designated nodes, and $A = \frac{DN}{N}$, the ratio between designated nodes and total nodes, in various stabilization degree S.

TABLE V
DESIGNATED NODES IN CHORD UNDER VARIOUS STABILIZATION DEGREE S

S	DN	A
0.33	14	0.00028
0.40	13	0.00026
0.80	4	0.00005

In some cases, there is no designated node to fail, as is the case where 20% of nodes fail in a Chord network with $S = 0.80$. In this case, Chord significantly reduce the number of failed lookups. The reason no designated node fails is because the probability to choose a designated node to fail or leave is also very small. With $S = 0.80$, the ratio between the number of designated nodes and the total number of nodes is $\frac{4}{50,000} = 8 \cdot 10^{-5}$. Assume we randomly choose $Z\%$ of nodes to fail or leave, the probability that a designated node fails or leaves is $\frac{8Z}{10^{-7}}$ (nearly zero) whereas the probability that an ordinary node fails or leaves is $\approx \frac{Z}{100}$.

In Chord, when a node leaves the system, it redistributes stored data to its successor. If its successor is not a designated node, the successor will become a designated node after the data have been redistributed. Hence, the numbers of designated nodes before and after the random simultaneous leaves are the same. However, for $S = 0.33$ and $S = 0.40$, some lookups still fail. This implies that the routing in Chord is not as robust as C-DGRID at lower S, because Chord cannot route lookup requests to the designated nodes. On the other hand, C-DGRID system incorporates the fact that there are many nodes that provide a certain resource type to increase its finger flexibility, which increases the reachability to the resource type.

V. RELATED WORKS

Routing-transferring model [11] and Iamnitchi et. al [9], [10] are decentralized information systems for grid, based on unstructured overlay network. Nodes periodically exchange resource information with each other. The resource information is the basis for the routing table at each node. The main issue with these schemes is lookups are not theoretically bounded. DGRID is based on DHT to enable efficient, scalable, and theoretically-bounded lookups.

Both XenoSearch [16] and self-organizing Condor pools [5] are decentralized information systems for grid, based on Pastry [14] – one of DHT algorithms. XenoSearch [16] stores grid-resource information on nodes arranged as Pastry network,

ignoring the relationship between data domain and node domain. Self-organizing Condor pools [5] uses Pastry to replicate resource information (advertisements) to physically-nearby location, subjected by time-to-live (TTL). Because replications are subjected to time-to-live (TTL), self-organizing Condor pool has a weaker result guarantee as pools may not be aware of some resource advertisements. DGRID ensures that resource information belongs to domain d is stored on a node from domain d, at the expense of increased maintenance cost of the overlay network. In addition, DGRID provides the same level of result guarantee as other DHTs whereby nodes can locate any existing resource information in the system.

SkipNet [8] is a structured overlay network that supports *content locality* where data can be explicitly stored at the desired node (e.g. data and node must be the same domain). However, content locality is not transparent since users need to enumerate domains when searching. To enforce data d to be stored at node n, SkipNet assigns an identifier to d such that $id(d) = id(n) \oplus key(a)$. A query for this data is formulated as $q = id(n) \oplus key(a)$. If a distributed information system for grid is implemented on top of SkipNet, then users need to enumerate existing domains in the grid if they require resources from many domains. With DGRID, controlled data placement is transparent to users; they need not enumerate domain names when doing lookups.

CFS [7] is a distributed storage where a file is stored in many servers; each server store one or more blocks of the file. CFS allows the virtualization of stronger servers to balance the load as 1) blocks may not occupy the identifier space in a uniformly distributed manner, which causes some servers to store zero blocks, and 2) stronger servers should store more blocks. DGRID automatically virtualizes each indexing server subjected to the number of resource types in the administrative domain. Assuming that a richer domain contains more type of resources, its indexing server can afford more storage to store data and can sustain the overhead of virtualization (i.e. more routing tables to be maintained). In addition, we believe that the overhead will not be very high as each domain has a limited number of resource types, especially we can devise a hierarchical naming scheme for resource type that properly maps to the identifier space.

VI. Conclusion

We have presented DGRID, a DHT-based indexing scheme for computational grid without data redistribution. DGRID increases security and availability, and improves performance by directing lookups only to administrative domains that owns the requested resources. Through simulations, we show that Chord-based DGRID compensates its higher overhead with smaller average path length (hops) and fewer number of failed lookups, even if a fraction of the network fails simultaneously. In addition, DGRID is more tolerant to the single point of failure of designated nodes. Finally, DGRID guarantees that no stale data are returned to clients.

As the future works, firstly, we will study the overhead of DGRID and investigate the performance of DGRID when looking up lots of resources with the same type. Then we will design parallel algorithms to look up many resources with the same type within a request. And we will also compare the performance of DGRID with other existing grid resource management approaches. We will also investigate efficient grid scheduling schemes within DGRID.

References

[1] *The Chord Project*, `http://www.pdos.lcs.mit.edu/chord/#downloads`

[2] *Globus Toolkit – Information Service*, `http://www.globus.org/mds/`

[3] *OGSI: Open Grid Services Infrastructure*, `https://forge.gridforum.org/projects/ogsi-wg`

[4] A. Andrzejak, Z. Xu, *Scalable, Efficient Range Queries for Grid Information Services*, Proceedings of the 2^{nd} International Conference on Peer-to-Peer Computing (P2P 2002), pp. 33–40, September 2002.

[5] A. R. Butt, R. Zhang, Y. C. Hu, *A Self-Organizing Flock of Condors*, Proceedings of the ACM/IEEE SC2003 Conference, pp. 42, November 2003.

[6] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*, Proceedings of the 4^{th} International Workshop on Job Scheduling Strategies for Parallel Processing (IPPS/SPDP'98), pp. 62–82, March 1998.

[7] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, *Wide-Area Cooperative Storage with CFS*, Proceedings of the $1i^{th}$ ACM Symposium on Operating Systems Principles (SOSP'01), pp. 202–215, October 2001.

[8] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, A. Wolman, *SkipNet: A Scalable Overlay Network with Practical Locality Properties*, Proceedings of the 4^{th} USENIX Symposium on Internet Technologies and and Systems (USITS'03), pp. 113–126, March 2003.

[9] A. Iamnitchi, *Resource Discovery in Large Resource-Sharing Environments*, Ph.D. Thesis, Department of Computer Science, The University of Chicago, December 2003.

[10] A. Iamnitchi, M. Ripeanu, I. Foster, *Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations*, Proceedings of the 1^{st} International Workshop on Peer-to-Peer Systems (IPTPS'02), pp. 232–241, March 2002.

[11] W. Li, Z. Xu, F. Dong, J. Zhang, *Grid Resource Discovery Based on a Routing-Transferring Model*, Proceedings of the 3^{rd} International Workshop on Grid Computing (GRID 2002), pp. 145–156, November 2002.

[12] Z. Németh, V. Sunderam, *Characterizing Grids: Attributes, Definitions, and Formalisms*, Journal of Grid Computing, 1(1):9–23, 2003.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, *A Scalable Content- Addressable Network*, Proceedings of ACM SIGCOMM 2001, pp. 161–172, August 2001.

[14] A. Rowstron, P. Druschel, *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems*, Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), pp. 329–350, November 2001.

[15] C. Schmidt, M. Parashar, *Flexible Information Discovery in Decentralized Distributed Systems*, Proceedings of the 12^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC-12), pp. 226–235, June 2003.

[16] D. Spence, T. Harris, *XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform*, Proceedings of the 12^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC-12), pp. 216–225, June 2003.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, Proceedings of ACM SIGCOMM 2001, pp. 149–160, August 2001.

[18] S. S. Vadhiyar, J. Dongara, *A Metascheduler for the Grid*, Proceedings of the 11^{th} IEEE Symposium on High Performance Distributed Computing (HDPC-11), pp. 343–351, July 2002.