DGRID: A DHT-Based Resource Indexing and Discovery Scheme for Computational Grids

Verdi March[1] **Yong Meng Teo**[1,2] **Xianbing Wang**[2]

[1]Department of Computer Science, National University of Singapore
[2]Singapore-MIT Alliance, National University of Singapore
3 Science Drive 2, Singapore 117543
Email: `teoym@comp.nus.edu.sg`

Abstract

Traditional DHT (Distributed Hash Tables) abstraction distributes data items among peer nodes on a structured overlay network. This introduces a number of issues when nodes are under different administrative authorities. In this paper, we propose DGRID, a new DHT abstraction for grid resource indexing and discovery where an administrative domain stores only its own data items. This is achieved by having each unique resource type belonging to an administrative domain to join a DHT as a node with a unique identifier. Using Chord as the underlying overlay graph, we show that DGRID lookup path length is at worst comparable with traditional DHT. However, DGRID is by design resilient to node failures without the need to replicate data items.

Keywords: grid, resource indexing and discovery, distributed hash table, data-item distribution.

1 Introduction

Resource discovery is an important infrastructure especially in a large computational grid consisting of compute resources distributed across administrative domains (Foster & Kesselman 1999). Typically, grid users search for specific resources before deploying their applications (Németh & Sunderam 2003). In its earlier stage, a centralized scheme supports resource discovery among a small number of administrative domains, where a central MDS (Czajkowski et. al. 2001) indexes all resources and processes all user queries. As the adoption of grid increases, the central MDS becomes a potential bottleneck and a single point of failure. Recently, there is growing interest in studying the use of DHT-based resource discovery for large computational grids. Instead of depending on a third-party central MDS, DHT-based schemes distribute queries across administrative domains organized as nodes in an overlay network (Cai et. al. 2004, Butt et. al. 2003, Spence & Harris 2003, Zhu et. al. 2004).

DHT (Gummadi et. al. 2003, Li et. al. 2004) is an approach to build large distributed systems that support efficient lookup with high result guarantee (Loo et. al. 2004). A typical DHT realizes the mapping of data items (i.e. resource metadata/pointer) to nodes (i.e. administrative domains) through a store operation. A unique *identifier* is associated with each node and a *key* is associated with each data item. The *identifier space* is partitioned among the nodes such that each node is responsible for storing all the data items whose key is in the node's portion of the space. By design, a node in the traditional DHT has no control over the distribution of its data items, and the number of data items belonging to others that it has to store (Daswani et. al. 2003). This characteristic, referred to as *data-item distribution*, introduces a number of new issues among interacting administrative domains when DHT is applied to facilitate resource discovery in computational grid. These issues include:

1. *Data Placement* A node has no control over the placement of its data items because (i) the mapping function considers only the distance between keys and nodes in the identifier space, and (ii) changes in overlay network cause data items to be remapped.

2. *Write Access* Each node is required to store data items belonging to other nodes. However, some organizations have strict storage policy, e.g. an organization's name server registers only domain names under the organization's administrative domain. The requirement of writable nodes also introduces a potential dependency on third party infrastructure that provides writable nodes to facilitate data-item distributions in DHT.

3. *Ownership* Commercial application requirements may not allow a node to store its data items (or even pointers to data items) on other nodes. Firstly, a node (owner) wants to ensure that it is the sole provider of its data items. For examples, some web sites do not allow their contents to be hosted or directly linked by other web sites, including search engines (WAN 2006, Reuters 2006). Secondly, when a data item is stored onto another node, the originating node of the data item does not want its data item to be used without its consent (Google 2006).

4. *Accountability* The management and accountability process for data items are spread across different administrative authorities, but supports for accounting vary among administrative domains. Furthermore, as a key is remapped, its accounting information scattered at different locations has to be consolidated. When the amount of accounting information is large, the consolidation increases network-bandwidth requirement.

The above issues, which also occur in current distributed systems such as world wide web, may hinder the commercial adoption of DHT-based scheme.

To address these issues, we propose a new approach called DGRID (DHT-based Grid Resource Indexing and Discovery) where an administrative domain stores only its own data items. On DGRID, each unique resource type in an administrative domain joins a DHT as a virtual node. The node identifier is derived by concatenating the key associated with the resource type, and the identifier assigned to the administrative domain. DGRID reuses the functionalities in existing structured overlay networks such as Chord (Stoica et. al. 2001) to leverage on the extensive research that have been done in structured overlay networks.

To illustrate our approach, we discuss the design of DGRID using Chord as the underlying overlay graph (Stoica et. al. 2001). Assume K denotes the number of unique resource types in a grid and N denotes the number of administrative domains. Our analysis shows that the lookup path length in DGRID is $O(\min(\log K, \log N))$, which is comparable with traditional DHT such as Chord. However, DGRID is by design resilient to node failures without the need to replicate data items as in conventional DHT. Our simulation results further confirm the theoretical lookup path length of DGRID and its resiliency to node failures.

The rest of this paper is organized as follows. Section 2 gives an overview of grid resource management and discusses the limitations of data-item distribution in computational grids. Section 3 presents the design of DGRID. Section 4 presents our theoretical analysis and section 5 discusses our simulation results. Section 6 compares DGRID to related work. Section 7 concludes this paper.

2 Background

We present an overview of distributed grid resource management, introduce the notations used in this paper, and discuss the limitations of applying traditional DHT such as Chord in computational grids.

2.1 Grid Resource Management

Assume a computational grid consisting of N administrative domains and K *resource types*. A resource type is a tuple of one or more attributes describing a resource such as $t = \{$`cpu`=“P4”, `memory`=“1 GB”$\}$. Figure 1 illustrates the anatomy of an administrative domain. Each administrative domain d has a set of resource types T_d, where each type $t \in T_d$ consists of many resource instances. Hence, $K = |\bigcup_{i=1}^{N} T_i|$. In current computational grid, resource metadata are published to an index server (e.g. MDS (Czajkowski et. al. 2001)) deployed by the administrative domain.

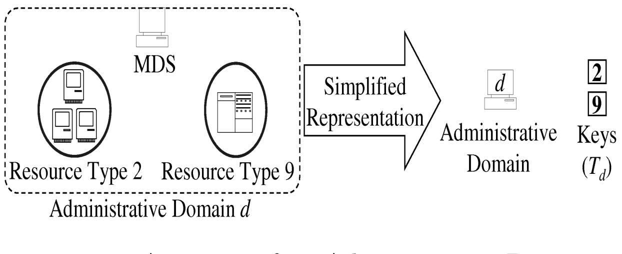
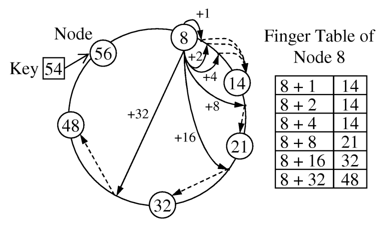


Figure 1: Anatomy of an Administrative Domain

To build a DHT-based system, we assign an m-bit identifier and an m-bit key to each administrative domain and resource type, respectively. The domain identifier is generated by hashing (e.g. SHA-1) the name of an administrative domain. Similarly, the key for a resource type is generated by applying SHA-1 on a single-attribute resource type, or Hilbert SFC on a multi-attribute resource type (Schmidt & Parashar 2003). Thus, T_d also denotes the set of unique keys in administrative domain d.

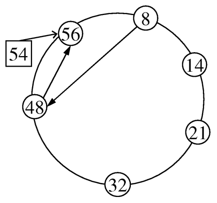
2.2 Limitations of Chord-based DHT

Chord (Stoica et. al. 2001) assigns an m-bit identifier to each node and organizes nodes as a ring. The ring represents a circular identifier space, and as a consequence, all arithmetic are modulo 2^m. In addition, Chord assigns an m-bit key to each data item and maps the key to the first node whose identifier is equal to or follows the key (Figure 2a). This node is called the successor of *key*, denoted by $successor(key)$. Consequently, the data item is also stored to $successor(key)$. Each node n maintains $O(m)$ fingers to speed-up the process of finding a successor. The i^{th} finger of n is $n.finger[i] = successor(n + 2^{i-1})$, where $1 \leq i \leq m$. Note that $n.finger[1]$ is also $successor(n)$.



8 + 1	14
8 + 2	14
8 + 4	14
8 + 8	21
8 + 16	32
8 + 32	48

(a) A Chord Ring Consisting of Six Nodes and One Key



(b) *lookup(54)* Initiated by Node 8

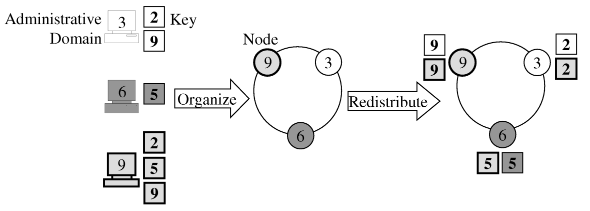
Figure 2: An Illustration of Chord

In a system with N nodes, Chord locates $successor(key)$ within $O(\log N)$ hops (see Theorem IV.2 in (Stoica et. al. 2001)). Intuitively, the process resembles a binary search where each step halves the distance to $successor(key)$. Each node n forwards a lookup request to the nearest known preceding node of the key. This is repeated until the request arrives at $predecessor(k)$, which will forward the request to $successor(key)$. Figure 2b illustrates the routing path from node 8 to $successor(54)$.

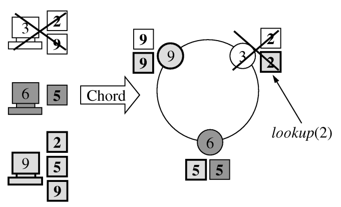
Traditional DHT such as Chord performs data-item distribution (Figure 3a). Apart from the issues discussed in Section 1, data-item distribution introduces stale data items and reduces the *resiliency* of DHT to node failures. In this paper, the term *resiliency* refers to the ability to locate existing resources whose originating administrative domain is still alive[1]. Figure 3b shows an example where administrative domain 3 fails. Though administrative domain 9 is still alive, a lookup for resource type 2 fails because Chord routes this request to node 3. Typically, traditional DHT replicates keys to improve its resiliency, e.g. replicates metadata of resource type 2 to node 6 (Figure 3c). However, this increases the

[1]In this paper, we do not focus on *availability*, i.e. resources are replicated such that DHT can still locate a resource despite that the master copy of the resource ceases. Since compute resources are by nature not replicable, availability is not a main topic of this paper.

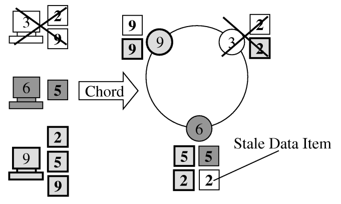
risk of stale data items, i.e. metadata pointing to unavailable resources. In the example, users can retrieve from node 6 metadata of resources in the inaccessible administrative domain 3.



(a) Distributing Data Items among Three Administrative Domains



(b) Administrative Domain 3 with Two Resource Types Fails



(c) Stale Data Items

Figure 3: Data-Item Distribution in Traditional DHT

Besides topology changes[2], stale data items also occurs when an administrative domain updates its resource metadata. In the example (Figure 3c), when administrative domain 9 updates its data item 5, the data item stored at administrative domain 6 becomes stale when the update has yet to be committed (i.e. the update from administrative domain 9 has yet to reach administrative domain 6).

3 Design of DGRID

This section presents the design of DGRID using Chord as the underlying overlay graph. We focus on the construction of DGRID overlay network and the routing of lookup requests.

3.1 Virtualization Scheme

Instead of distributing data items, DGRID maps each key onto its own administrative domain by virtualizes an administrative domain into DGRID nodes. Figure 4 illustrates the virtualization of three administrative domains. Let d denotes an administrative domain and T_d denotes a set of keys (i.e. resource types) owned by d. To map key $t \in T_d$ onto d, t joins DGRID as node $n_{t,d}$. Thus, administrative domain d is virtualized into $|T_d|$ nodes. The *node identifier* of node $n_{t,d}$ is $t|d$, which is the concatenation of key t as the prefix and the identifier of d as the suffix[3]. By combining t and d as a node identifier, DGRID allows several administrative domains to own the same key t; this is an important requirement for computational grids where two or more administrative domains share the same type of compute resources.

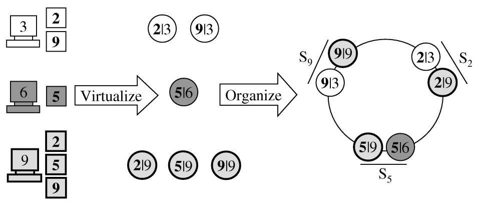


Figure 4: Virtualizing Three Administrative Domains into Six Nodes

Our scheme divides the DGRID overlay into *segments*. Each segment S_t consists of nodes whose identifier is prefixed with t. Thus, segment S_t represents resources of the same type t but belonging to different administrative domains. Segment indexing improves DGRID resiliency to node failure and reduces lookup path length. Figure 4 shows a ring overlay with three segments.

The main features of DGRID are:

1. *No stale data items*
   There are no stale data items when the overlay topology changes, or when an administrative domain updates its data items.

2. *Resilient to node failures*
   A DGRID segment S_t represents resources of type t shared by different administrative domains. This increases the probability of successful lookup in the event of node failures even if resources are not replicated.

3.2 Lookup Optimization

DGRID supports flat naming scheme in its lookup interface, i.e. *lookup(t)*. The flat naming scheme abstracts away the originating administrative domain of resources. This supports queries formulated as "find resource type t", as opposed to hierarchical naming scheme which requires users to know the originating administrative domain of resources (i.e. "find resource type t from administrative domain d")[4].

The lookup algorithm, as shown in Figure 5, returns *any* administrative domain in segment S_t. A lookup request is forwarded from one segment to another segment (line 4, 8, and 18 in Figure 5a) such that each routing step halves the distance, in term of segments, to S_t. *Routing by segment* in DGRID reduces lookup path length to $O(\log K)$, and is equivalent to prefix-based routing even though the underlying Chord protocol does not support prefix-based routing.

Our lookup algorithm also features *shared finger tables*. With this optimization, the DGRID lookup algorithm considers all the $|T_d|$ finger tables maintained by each administrative domain d (line 3 and 13 in Figure 5a, and line 3 and 14 in Figure 5b). Thus, all finger tables within an administrative domain are shared to limit DGRID lookup path length to at most $O(\log N)$ hops as in Chord.

Figure 6 illustrates the routing path of *lookup(3)* initiated by node 0, assuming m = 2-bit and each administrative domain has one resource type. Since node 1 is not in segment S_3 ($\mathbf{11}_2$), it forwards *lookup(3)* to any node in the nearest preceding segment, which is node 9 ($\mathbf{10}01_2$) in segment S_2 ($\mathbf{10}_2$). Then, node 9 forwards the lookup request to node 15 in segment S_3.

[2]Due to node joins, leaves, or fails.

[3]Similar to node identifiers in traditional DHT, the identifier of an administrative domain is derived by hashing the domain name or is obtained from a certification authority (Castro et. al. 2002).

[4]Such queries are reminiscent of HTTP requests: "retrieve document `index.html` from site `www.comp.nus.edu.sg`".

```
//Ask $d$ to find resource type $t$
$d$.**lookup**($t$)
    **for each** $j \in T_d$ **do**
        **if** $j == t$ **then**
            //$d$ is in $S_t$
            **return** $d$;

    $n = find\_segment\_in\_fingers(t)$;
    **if** $n \neq$ NOT_FOUND **then**
        //$h$ is in the preceding segment of $S_t$
        **return** $suffix(n)$;

    **for each** $j \in T_d$ **do**
        $n = j|d$;
        **if** $n < t|0 < n.successor$ **then**
            **return** NOT_FOUND;

    $n = closest\_preceding\_segment(t)$;
    **return** $n.lookup(t)$;
```

(a) Main Algorithm

```
//Ask $d$ to find a finger pointing to $S_t$
$d$.**find_segment_in_fingers**($t$)
    **for each** $j \in T_d$ **do**
        $n = j|d$;
        **for** $i = 1$ **to** $m$ **do**
            **if** $prefix(n.finger[i]) == k$ **then**
                **return** $n.finger[i]$;

    **return** NOT_FOUND
```

```
//Ask $d$ to find the closest predecessor of $t$.
$d$.**closest_preceding_node**($t$)
    $id = t|0$;
    $x = id + 1$;  //Furthest predecessor
    **for each** $t \in T_d$ **do**
        $n = t|d$;
        **for** $i = m$ **downto** $1$ **do**
            **if** $(n < n.finger[i] < id)$
                    **and** $(x < n.finger[i] < id)$ **then**
                $x = f$;

    **return** $x$;
```

(b) Helper Functions

Figure 5: DGRID Lookup

The segment-based overlay graph improves *finger flexibility*[5] whereby $n.finger[i]$ is allowed to point to any nodes in the same segment as $successor(n+2^{i-1})$. This is an improvement over Chord's finger flexibility which is $O(1)$. DGRID exploits finger flexibility to improve the robustness of lookups by maintaining, in every node, backups for each of its fingers[6]. Therefore, if finger f dies, n still has a pointer to another node in the same segment as f. A simple scheme to maintain backup fingers is to piggyback periodic finger corrections (see (Stoica et. al. 2001) for detail). Thus, in addition to returning $successor(f)$ when correcting finger f, we also piggyback fingers prefixed by $prefix(f)$ kept in the finger table of $successor(f)$.

4 Analysis

In this section, we analyze and compare the costs of virtualization and lookup in DGRID with traditional Chord-based DHT, hereafter referred to as Chord. Chord virtualizes N administrative domains into N nodes, one node per administrative domain. In contrast, DGRID virtualizes the N administrative domains, each consisting of $|T_d|$ resource types, into $V = |T_d|N$ nodes. As in (Stoica et. al. 2001), these costs hold "with high probability".

[5]The amount of freedom available when choosing a finger (Gummadi et. al. 2003).

[6]To reduce lookup latency, DGRID may further exploit segment indexing to support proximity-based routing.

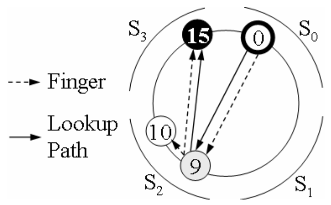


Figure 6: Example of *lookup()*

4.1 Lookup

In this section, we analyze the lookup path length of DGRID.

Lemma 1. *The probability that an administrative domain d owns a resource type t is $P(t \in T_d) = \ln \frac{K}{K-|T_d|}$, with high probability.*

Proof. Since T_d is the set of resource types in d, the probability of $t \in T_d$ is:

$$P(t \in T_d) = P(e_1) + \ldots + P(e_{|T_d|}|\overline{e_1}, ..., \overline{e_{|T_d|-1}})$$
$$= \sum_{i=1}^{|T_d|} P(e_i|\overline{e_1^{i-1}})$$

where e_i denotes the event for $t_i = t$, and $\overline{e_i}$ denotes the event for $t_i \neq t$. If $|T_d| > 1$, then assuming $|T_d| \ll K$, we can use the first-order estimation on $P(t \in T_d)$.

$$P(t \in T_d) = \begin{cases} 0 & \text{if } |T_d| = 0 \\ \frac{1}{K} & \text{if } |T_d| = 1 \\ \sum_{i=K-|T_d|+1}^{K} \frac{1}{i} & \end{cases}$$

Since $H_x = \sum_{i=1}^{x} \frac{1}{i} = \ln x + O(1)$, then

$$P(t \in T_d) = \begin{cases} 0 & \text{if } |T_d| = 0 \\ \frac{1}{K} & \text{if } |T_d| = 1 \\ H_K - H_{K-|T_d|} & \end{cases}$$
$$= \begin{cases} 0 & \text{if } |T_d| = 0 \\ \frac{1}{K} & \text{if } |T_d| = 1 \\ ln \ \frac{K}{K-|T_d|} & \end{cases}$$

■

Lemma 2. *Routing by segments leads to $O(\log K)$-hops lookup path length.*

Proof. In Chord, the distance between n and $n.finger[i+1]$ is twice the distance between n and $n.finger[i]$, and the largest finger points to $successor(N/2)$. Chord routes a lookup request from one node to another, and each step halves the distance to the destination node. Based on Theorem IV.2 (Stoica et. al. 2001), this costs $O(\log N)$.

We now show the similarity of finger tables in DGRID and Chord. Let S denotes the average number of nodes in a segment, which is $O(NP(k \in T_d))$. In DGRID, each node n maintains $O(\log V)$ unique fingers. The first $O(\log S)$ fingers points to the segment containing the successor of n. The remaining fingers, which amount to $O(\log V - \log S)$, points to $O(\log V - \log S)$ different segments because the distance between n and $n.finger[j+1+\log S]$ is twice the distance between n and $n.finger[j + \log S]$. In addition, the largest finger of n points to $successor(N/2)$, which is also the succeeding segment of $K/2$. Hence, DGRID routes a lookup request from one segment to another and each hop halves the distance, *in term of segments*, to the destination segment. By the same argument as in Chord, a lookup in a K-segments DGRID costs $O(\log K)$ hops. ■

Theorem 1. *The cost of a DGRID lookup is $O(\min(\log K, \log N))$. For $V \geq N$, the lookup performance of DGRID would not be worse than $O(\log N)$ in Chord.*

Proof. When $K \leq N$, $\log K \leq \log N$ and according to Lemma 2, this theorem is true.

Now consider $K > N$. Due to shared finger tables, visiting administrative domain d is equivalent to visiting T_d nodes. Thus, when administrative domain d forwards a *lookup(t)* request, d chooses $s, u \in T_d$ such that $s < t < u$, but there must be no $v \in T_d$ such that $s < v < u$. The distance between $n_{s,d}$ and $n_{t,d}$ is $K/|T_d|$; this is also the maximum distance between any two segments. Since $K \leq V$, then $K/|T_d| = O(V/|T_d|) = O(N)$. Thus, from $n_{h,d}$, *lookup(t)* can be routed to segment S_t within $O(\log N)$ steps. ■

Theorem 1 shows that the lookup path length in DGRID is at worst equal to the lookup path length in Chord. Due to the shared finger tables, the number of hops to reach a certain node is affected the number of hosts in the physical network (N) instead of the number of nodes in the overlay network (V).

4.2 Virtualization

The following theorems show the properties of DGRID in terms of cost of virtualization, number of fingers per host, cost of updating data items, and overhead of replication.

Theorem 2. *In a computational grid with N administrative domains, virtualizing an administrative domain in DGRID and in Chord costs $O(|T_d| \log^2 V)$ and $O(\log^2 N + |T_d| \log N + K \ln \frac{K}{K-|T_d|})$, respectively.*

Proof. DGRID virtualizes an administrative domain into $|T_d|$ nodes in an overlay graph of size V. Since the cost of each join is $O(\log^2 V)$, the total cost is $O(|T_d| \log^2 V)$.

In Chord, virtualization of an administrative domain consists of an administration-domain join, $|T_d|$ data-item stores, and migration of some existing data items. A join costs $O(\log^2 N)$ and each store operation costs $O(\log N)$. The migration process moves $O(KP(t \in T_d))$ data items because it affects $\frac{K}{N}$ resource types (unique keys) and there are $NP(t \in T_d)$ data items per resource type. Thus, the total cost is $O(\log^2 N + |T_d| \log N + K \ln \frac{K}{K-|T_d|})$. ■

Theorem 3. *In DGRID, each administrative domain maintains $O(|T_d| \log V)$ fingers.*

Proof. Since DGRID virtualizes each administrative domain into $|T_d|$ nodes and each node maintains $O(\log V)$ fingers, the administrative domain maintains $O(|T_d| \log V)$ fingers in total. ■

A higher number of fingers increases implies a higher overhead in maintaining an overlay graph. This affects the scalability of DGRID particularly when an administrative domain is virtualized into a high number of nodes. To reduce the overhead of *periodic stabilizations*, i.e. a mechanism whereby each node periodically invoke a finger-correction procedure, nodes need not to correct all their fingers each time they invoke the finger-correction procedure. Instead, each invocation corrects only a subset of a node's fingers, e.g. the successor pointer and another randomly-chosen finger; this is similar to Chord's implementation of periodic stabilizations.

Theorem 4. *In DGRID, the finger flexibility is $O(N \ln \frac{K}{K-|T_d|})$. In Chord, the finger flexibility is $O(1)$.*

Proof. In DGRID, the i^{th} finger of node n can be any node with the same prefix as $successor(n+2^{i-1})$. The number of administrative domains that own resource type t is $NP(t \in T_d)$. Thus, there are $O(NP(t \in T_d)) = O(N \ln \frac{K}{K-|T_d|})$ nodes prefixed by t.

In Chord, the i^{th} finger of n must be $successor(n+2^{i-1})$, and hence, $O(1)$ finger flexibility. ■

As mentioned in Section 3.2, a higher finger flexibility increases the robustness of lookup. It further allows optimizations such as proximity-based routing to reduces lookup latency[7].

Theorem 5. *Adding a resource type in Chord costs $O(\log N)$ with high probability. In DGRID, adding a resource type that already exist in the administrative domain costs only $O(1)$, and adding a* new *resource type to an administrative domain costs $O(\log^2 V)$ with high probability.*

Proof. Adding a resource in Chord involves storing a data item; this costs $O(\log N)$ hops.

In DGRID, if $t \in T_d$, no new node is created and hence the cost is $O(1)$. However, if $t \notin T_d$, a new node is created and joins the system with a cost of $O(\log^2 V)$. ■

In applications such as P2P file sharing, sharing a new file is equal to adding a new resource type. However, in computational grid, a resource type consists of many resource instances, and an administrative domain can add new instances to one of its existing resource type. Theorem 5 shows that using DGRID, the administrative domain does not need to notify other nodes in the DGRID overlay.

Theorem 6. *In DGRID, the total number of data items with the same resource type is $O(N \ln \frac{K}{K-|T_d|})$ with high probability. In Chord, if each data items is replicated $O(\log N)$ times, then the number of data items with the same resource type is $O(N \ln \frac{K}{K-|T_d|} \log N)$.*

Proof. There are $O(NP(t \in T_d))$ data items with the same key. Hence, in DGRID there are $O(N \ln \frac{K}{K-|T_d|})$ data items with resource type t. In Chord, each data items is replicated $\log N$ times. Hence, the total number of data items with resource type t is $O(N \ln \frac{K}{K-|T_d|} \log N)$. ■

DGRID does not need to replicate data items to improve the lookup resiliency to node failures. This eliminates the network bandwidth required to replicate data items and furthermore, avoids the complexity in maintaining consistency among replicas.

[7]The current implementation of DGRID has yet to exploit this optimization.

4.3 Summary

In this section, we analyze the lookup performance and the maintenance overhead of DGRID. We show that the lookup path length in DGRID is the same as traditional DHT even though DGRID virtualizes administrative domains into nodes. The overhead to maintain a DGRID overlay is higher because the virtualization increases the size of the overlay. Compared to traditional DHT, each administrative domain in DGRID has more fingers to correct. Consequently, the scalability of DGRID is determined by the number of nodes associated with each administrative domain. When each administrative domain is virtualized into one node only, the scalability of DGRID is equal to traditional DHT.

5 Simulation Results

To study the lookup performance and resiliency of DGRID, we conducted simulation experiments based on the Chord simulator used in (Stoica et. al. 2001). We simulated 500,000 lookup requests (Poisson distribution with 1 second mean arrival rate) and each request consists of a randomly selected data item and is initiated by a randomly chosen node. We set m = 20-bit, the network latency between administrative domains is 50 ms (exponentially distributed), and the overhead of request processing is [5, 15] ms (uniformly distributed). In our simulation, we measure the lookup performance by the number of hops in the overlay network. We do not consider the lookup latency due to routings on the physical network infrastructure.

5.1 Lookup Performance

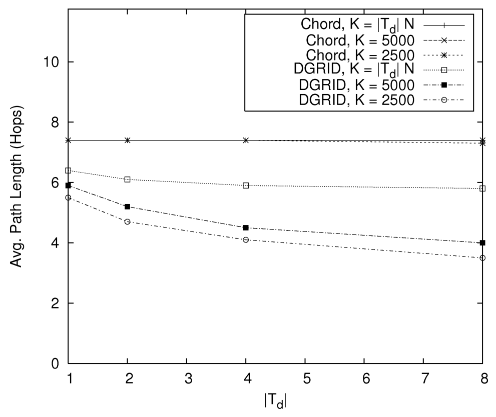
In this section, we evaluate the lookup path length in DGRID and Chord. We assume a computational grid consisting of N administration domains, where N is varied from 10,000 to 25,000. Each administrative domain has $|T_d|$ resource types on average, i.e. the number of resource types per administrative domain is in the range of $[0.5|T_d|, 1.5|T_d|]$ (uniformly distributed).

As shown in Figure 7, the average lookup path length in DGRID is 20-30% lower than in Chord. In Chord, the average lookup path length is $O(\log N)$. In DGRID, according to Theorem 1, when $(K = |T_d|N) > N$, the average lookup path length is affected only by N, and hence, increasing K does not increase the average lookup path length. However, for $K \leq N$, the average lookup path length increases with K.

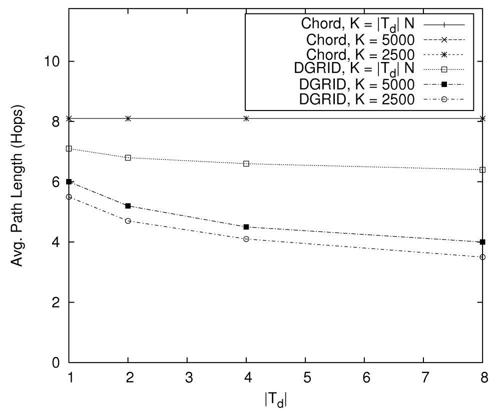
In DGRID, increasing $|T_d|$ reduces the average path length due to the following reasons. Based on Corollary 3, an increased in $|T_d|$ also increases the number of fingers per administrative domain. Studies in (Gupta et. al. 2004, Rodrigues & Blake 2004) also reveal that maintaining more fingers reduces the lookup path length. Secondly, an increased in $|T_d|$ increases the number of segments occupied by an administrative domain, and hence, each administrative domain has a higher probability to be visited.

5.2 Resilience to Simultaneous Node Failures

To evaluate impact of data-item distribution to the lookup resiliency, we failed a fraction of administrative domains (20% to 40%) and we measure the percentage of failed lookups. We exploit the property of finger flexibility in DGRID by setting four backups per finger.



(a) N = 10,000



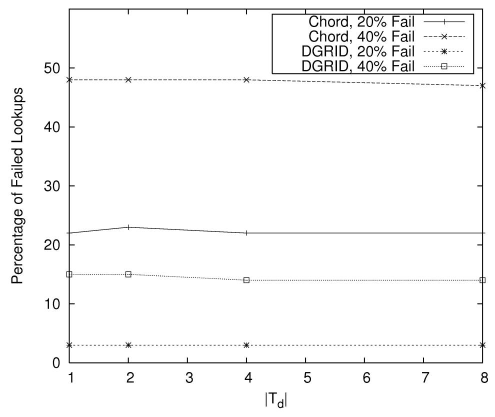
(b) N = 25,000

Figure 7: Average Lookup Path Length

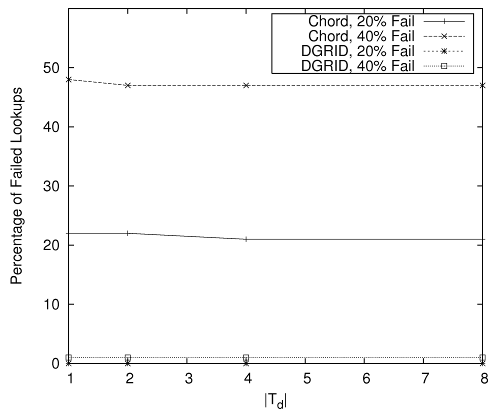
In terms of failed lookup requests, Figure 8 shows that for $K > N$ and $K \leq N$, DGRID is 70% and 95% lower than in Chord, respectively. For $K > N$, in Chord, data items from an administrative domain are distributed and stored in other administrative domains. In the event of the administrative domain that stores the data item fails or leaves the system, a lookup request to that administrative domain will be unsuccessful though the administrative domain that owns the data item is still alive. For $K \leq N$ (Figure 8b), the number of failed lookups in DGRID is 95% lower than in Chord because DGRID exploits the property that a resource type t is available in $NP(t \in T_d)$ administrative domains, a reasonable assumption in computational grid. Hence, even if some of these domains fail, DGRID can still reach the remaining administration domains through the backup fingers. Thus, even though by design DGRID does not replicate data items, it is resilient to node failures. On the other hand, traditional DHT has a lower resiliency when data items are not replicated.

5.3 Summary

The simulation results confirm our theoretical analysis on DGRID lookup performance. We verify Theorem 1 that the lookup path length of Chord-based DGRID is at most equal to traditional Chord. We also verify that DGRID achieves better resiliency than Chord. Although data items (i.e. resource metadata) are not replicated, the number of failed lookups in DGRID is lower because each administrative domain is responsible only for its own resource metadata and lookup requests are routed to administrative domains that share the requested resources.



(a) $K = |T_d|N$



(b) $K = 5{,}000$

Figure 8: Percentage of Failed Lookups for $N =$ 25,000

6 Related Work

The routing-transferring model (Li et. al. 2002) and the scheme proposed by Iamnitchi (Iamnitchi et. al. 2002) are examples of grid information systems based on unstructured overlay networks. Replication of resource information to all nodes proposed in (Li et. al. 2002) consumes communication bandwidth. Iamnitchi proposes to replicate information based on the small-world effect and uses heuristics to aid lookup. However, heuristics do not guarantee that a lookup will successfully find resources. DGRID, based on DHT, provides stronger lookup guarantee and scalable lookup performance.

MAAN (Cai et. al. 2004), self-organizing Condor pools (Butt et. al. 2003), XenoSearch (Spence & Harris 2003), and RIC (Zhu et. al. 2004) are examples of grid information systems based on DHT. While DGRID is also a DHT-based approach, it does not distribute data items. As a result, DGRID does not introduce stale data items when nodes fail or leave, and it achieves better resiliency without replicating data items.

SkipGraph (Aspnes & Shah 2003) also supports the no-store scheme by associating a resource type with a node and organizing nodes as a distributed data structure resembling a skip list. DGRID generalizes SkipGraph by first, allowing a resource type to be shared by different administrative domains. Secondly, DGRID can be implemented using different flavors of DHT, as long as the chosen DHT does not dynamically modify node identifiers.

SkipNet (Harvey et. al. 2003) supports *content locality* to store a data item on a specific node. Compared to our proposed scheme, SkipNet provides greater flexibility for an administrative domain to decide where its data items are stored. However, SkipNet uses a hierarchical naming scheme which requires users to identify the originating administrative domain of resources; this is similar with how users locate web content. As an example, *lookup*(`domain.com/r`) facilitates users to access resource `r` shared only by `domain.com`. On the other hand, DGRID supports flat naming scheme which facilitates hash-table-like lookups, e.g. *lookup*(`r`), to locate all administrative domains that share resource `r`.

7 Conclusion

We have presented DGRID, a DHT-based resource indexing and discovery scheme for computational grids. In contrast to traditional DHT abstraction, there is no data-item distribution in DGRID. The main advantages are (i) increased autonomy as each administrative domain stores only its own data items, and (ii) resiliency to node failures because a lookup request can be satisfied by any nodes in a DGRID segment. Using Chord as the underlying overlay graph, our analysis shows that for a computational grid with N administrative domains and K unique resource types, the lookup performance of $O(\min(\log K, \log N))$ in DGRID is at worst comparable with traditional DHTs. Simulation results confirm our theoretical analysis on DGRID lookup path length and further show that DGRID does not need to rely on data-item replications to improve its lookup resiliency to node failures.

A deficiency of DGRID is the higher cost in maintaining a larger overlay network (Theorem 2 and Theorem 3). We are addressing this issue in two directions. Firstly, we propose to collapse nodes within a segment into a second-level overlay network (March et. al. 2005). The new two-level hierarchical DGRID with smaller subgraphs facilitates more efficient overlay-network maintenance. Secondly, we are investigating a new scheme where each administrative domain adaptively adjusts the number of fingers maintained. In this scheme, each administrative domain approximate the size of the overlay network to determine the minimum number finger required in order to support robust lookup with short lookup path length.

We are enhancing DGRID to support applications in a hybrid system where data-item distribution is restricted to reserved DGRID segments; this facilitates transparent replications to a set of publicly writable nodes. Selective replication will also address the load imbalance problem where all lookups for a frequently-requested data item are routed to its originating domain. To incorporate selective replication into DGRID, an administrative domain who is willing to store data items belonging to other administrative domains joins DGRID as one node only. The node identifier is prefixed by a reserved key.

Lastly, we are extending DGRID to support multi-attribute resource indexing and discovery using space-filling curve (March & Teo 2006). Using space-filling curve, a resources described by many attributes are indexed by being assigned a key drawn from a one-dimensional identifier space. Similarly, a multi-attribute range query is transformed into many search keys through the space-filling curve; we then process these search keys with one or more DGRID lookups.

References

Aspnes, J. & Shah, G. (2003), Skip Graphs, *in* 'Proc. of the 14th Annual ACM-SIAM Symp. on Discrete Algorithms', Baltimore, Maryland, USA, pp. 384–393.

Butt, A. R., Zhang, R. & Hu, Y. C. (2003), A self-organizing flock of Condors, *in* ‘Proc. of the ACM/IEEE SC2003 Conf.’, Phoenix, AZ, USA, pp. 42.

Cai, M., Frank, M., Chen, J. & Szekely, P. (2004), ‘MAAN: A Multi-Attribute Addressable Network for grid information services’, *Journal of Grid Computing* **2**(1), 3–14.

Castro, M., Druschel, P., Ganesh, A., Rowstron, A., & Wallach, D. S. (2002), Secure Routing for Structured Peer-to-Peer Overlay Networks, *in* ‘Proc. of the 5th Usenix Symp. on Operating Systems Design and Implementation (OSDI 2002)’, Boston, Massachusetts, USA, pp. 299–314.

Czajkowski, K., Fitzgerald, S., Foster, I. & Kesselman, C. (2001), ‘Grid Information Services for Distributed Resource Sharing’, *‘Proc. of the 10th IEEE International Symp. on High Performance Distributed Computing (HPDC-10)’*, San Francisco, CA, USA, pp. 181–194.

Daswani, N., Garcia-Molina, H. & Yang, B. (2003), Open problems in data-sharing peer-to-peer systems, *in* ‘Proc. of the 9th International Conf. on Database Theory (ICDT 2003)’, Siena, Italy, pp. 1–15.

Foster, I. & Kesselman, C., eds (1999), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers.

Google (2006), ‘Google’s opposition to the government’s motion to compel’, `http://googleblog.blogspot.com/pdf/Google_Oppo_to_Motion.pdf`.

Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S. & Stoica, I. (2003), The impact of DHT routing geometry on resilience and proximity, *in* ‘Proc. of ACM SIGCOMM 2003’, Karlsruhe, Germany, pp. 381–394.

Gupta, A., Liskov, B. & Rodrigues, R. (2004), Efficient routing for peer-to-peer overlays, *in* ‘Proc. of 1st Symp. on Networked Systems Design and Implementation (NSDI’04)’, San Francisco, California, USA, pp. 113–126.

Harvey, N. J. A., Jones, M. B., Saroiu, S., Theimer, M. & Wolman, A. (2003), SkipNet: A scalable overlay network with practical locality properties, *in* ‘Proc. of the 4th USENIX Symp. on Internet Technologies and Systems (USITS’03)’, Seattle, Washington, USA, pp. 113–126.

Iamnitchi, A., Ripeanu, M. & Foster, I. (2002), Locating data in (small-world?) peer-to-peer scientific collaborations, *in* ‘Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS’02)’, Cambridge, MA, USA, pp. 232–241.

Li, J., Stribling, J., Gil, T. M., Morris, R. & Kaashoek, M. F. (2004), Comparing the performance of distributed hash tables under churn, *in* ‘Proc. of the 3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS’04)’, La Jolla, CA, USA, pp. 87-99.

Li, W., Xu, Z., Dong, F. & Zhang, J. (2002), Grid resource discovery based on a routing-transferring model, *in* ‘Proc. of the 3rd Intl. Workshop on Grid Computing (GRID 2002)’, Baltimore, MD, USA, pp. 145–156.

Loo, B. T., Huebsch, R., Stoica, I. & Hellerstein, J. M. (2004), The case for a hybrid P2P search infrastructure, *in* ‘Proc. of the 3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS’04)’, La Jolla, CA, USA, pp. 141–150.

March, V. & Teo, Y. M. (2005), Multi-attribute range queries in read-only DHT, *in* ‘Proc. of the 15th IEEE Conf. on Computer Communications and Networks (ICCCN 2006)’, Arlington, Virginia, USA, pp. 419–424.

March, V., Teo, Y. M., Lim, H. B., Eriksson, P. & Ayani, R. (2005), Collision detection and resolution in hierarchical peer-to-peer systems, *in* ‘Proc. of the 30th IEEE Conf. on Local Computer Networks (LCN 2005)’, Sydney, Australia, pp. 2–9.

Németh, Z. & Sunderam, V. (2003), ‘Characterizing grids: Attributes, definitions, and formalisms’, *Journal of Grid Computing* **1**(1), 9–23.

WAN (2006), ‘Newspaper, magazine and book publishers organizations to address search engine practices’, `http://www.wan-press.org/article9055.html`.

Reuters (2006), ‘WPP’s Sorrell sees Google as threat, opportunity’, `http://today.reuters.com/news/articlebusiness.aspx?type=media&storyid=nN01402884&imageid=&cap=`.

Rodrigues, R. & Blake, C. (2004), When multi-hop peer-to-peer lookup matters, *in* ‘Proc. of the 3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS’04)’, La Jolla, CA, USA, pp. 112–122.

Schmidt, C. & Parashar, M. (2003), Flexible information discovery in decentralized distributed systems, *in* ‘Proc. of the 12th IEEE International Symp. on High Performance Distributed Computing (HPDC-12)’, Seattle, WA, USA, pp. 226–235.

Spence, D. & Harris, T. (2003), XenoSearch: Distributed resource discovery in the XenoServer open platform, *in* ‘Proc. of the 12th IEEE International Symp. on High Performance Distributed Computing (HPDC-12)’, Seattle, WA, USA, pp. 216–225.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. & Balakrishnan, H. (2001), Chord: A scalable peer-to-peer lookup service for internet applications, *in* ‘Proc. of ACM SIGCOMM 2001’, San Diego, CA, USA, pp. 149–160.

Zhu, C., Liu, Z., Zhang, W., Xiao, W., Xu, Z. & Yang, D. (2004), ‘Decentralized grid resource discovery based on resource information community’, *Journal of Grid Computing* **2**(3), 261–277.