Distributed Aggregation Algorithms with Load-Balancing for Scalable Grid Resource Monitoring

Min Cai and Kai Hwang

University of Southern California
Dept. of Computer Science
Los Angeles, CA 90089 USA
{mincai, kaihwang}@usc.edu

Abstract

Scalable resource monitoring and discovery are essential to the planet-scale infrastructures such as Grids and PlanetLab. This paper proposes a scalable Grid monitoring architecture that builds distributed aggregation trees (DAT) on a structured P2P network like Chord. By leveraging Chord topology and routing mechanisms, the DAT trees are implicitly constructed from native Chord routing paths without membership maintenance. To balance the DAT trees, we propose a balanced routing algorithm on Chord that dynamically selects the parent of a node from its finger nodes by its distance to the root.

This paper shows that this balanced routing algorithm enables the construction of almost completely balanced DATs, when nodes are evenly distributed in the Chord identifier space. We have evaluated the performance and scalability of a DAT prototype implementation with up to 8192 nodes. Our experimental results show that the balanced DAT scheme scales well to a large number of nodes and corresponding aggregation trees. Without maintaining explicit parent-child membership, it has very low overhead during node arrival and departure. We demonstrate that the DAT scheme performs well in Grid resource monitoring.

1. Introduction

Scalable resource monitoring and discovery are essential to the planet-scale infrastructures such as Grids[10] and PlanetLab[5]. In these distributed environments, administrators need to continuously monitor some global system properties for capacity planning or system diagnostics. Users or applications need to monitor the real-time status of resources, and to discover the appropriate ones that are of their interests. However, resource monitoring and discovery in Grids are quite challenging due to their increasing scales. For example, the current PlanetLab consists of 706 machines at 340 sites[5], and the planet-scale Grid will have 100,000 CPUs in 2008[19]. P2P Grid such as the SETI @Home[11] achieves massively distributed computing by aggregating CPU cycles from millions of contributing computers.

Most existing systems in Grids maintain a centralized server [3][8][15] or a set of hierarchically organized servers[9][13] to aggregate and index resource information. For example, R-GMA[8], GridRM[3] and CoMon[15] use a centralized server to monitor all resource information. In contrast, Globus MDS2[9] and Ganglia[13] employ a set of hierarchical servers, such as LDAP-based directory server. The centralized server might become both a bottleneck and a single point of failure in a planet-scale environment. Zhang et al[23] show that GIIS in MDS2 and Manager in Hawkeye can only manage up to 100 GRIS or Agent servers. On this scale, both GIIS and GRIS need to enable data caching with large time-to-live (TTL) values, which is not suitable for real-time status such as CPU load. In addition, the partitioning scheme in hierarchical systems is often predefined and can not adapt to the dynamic change of Grid environments. For example, if the upper level GIIS fails, the low level GRIS needs to be manually redirected.

To overcome the above shortcomings, several peer-to-peer (P2P) schemes, e.g. MAAN[6], NodeWiz[4] and SWORD[14], have been proposed to index and discover Grid resources in a structured P2P network. By using appropriate routing schemes, search queries are routed to the nodes that are responsible for indexing the corresponding resources. Therefore, these schemes scale well to large number of participating nodes. On the other hand, their flat indexing structures pose a major challenge to the global resource monitoring in Grids due to its large-scale and decentralized nature.

Distributed aggregation is an essential building block for global resource monitoring in large-scale Grids. By employing a *distributed aggregation tree* (DAT), the global resource status can be calculated by recursively applying an aggregate function on a subset of local status. A distributed aggregation scheme has to meet three requirements on scalability, adaptiveness, and load balance. First, to scale to a large number of nodes, each aggregation should only introduce a limited number of messages with respect to the network size. The DAT tree should have low construction

This work was supported by NSF Grant ITR-0325409 at the University of Southern California. Corresponding author is Kai Hwang. Tel: 213-740-4470, and Fax: 213-740-4418.

and maintenance overhead. Second, the aggregation scheme has to adapt to the dynamics of node arrival and departure. Third, the aggregation workload should be distributed evenly among all nodes without any performance bottleneck. Load balancing is thus essential for both workload fairness and system scalability.

This paper proposes a P2P-based architecture for Grid resource monitoring and discovery. Our scheme extends P2P-based Grid resource discovery with DAT trees for global resource monitoring. The DAT trees are constructed among nodes by leveraging a structured P2P network, i.e. Chord[18]. In DAT, all nodes use a *balanced routing* scheme to build a balanced DAT tree towards the root node. We have implemented a prototype DAT system running on top of RPC protocol or on a discrete event simulation engine. We evaluated the performance of the DAT system on Grid resource monitoring with up to 8192 nodes.

The remainder of this paper is organized as follows: Sec.2 describes the P2P-based Grid resource monitoring architecture. We present the DAT construction algorithms in Sec. 3 and a prototype implementation in Sec. 4. The performance results of the DAT system are reported in Sec. 5. We discuss the related work in Sec. 6 and conclude this paper in Sec. 7.

2. P2P-based Grid Monitoring Architecture

In this section, we present a scalable P2P-based Grid monitoring architecture. This architecture leverages a multi-attribute addressable network for indexing resources and a distributed aggregation tree for summarizing global resource information.

2.1. Architecture

The P-GMA architecture extends the Grid Monitoring Architecture (GMA) proposed by the Global Grid Forum [20] with two important components, i.e. P2P-based resource indexing and aggregation. Fig. 1 shows the layered architecture of P-GMA. The seven layers in P-GMA are sensor, producer, indexing, aggregation and consumer layers. As suggested by Zanikolas and Sakellariou[22], a sensor monitors the status of one or more resources and generates events to producers. The sensor could be simply some scripts that collect the system status from the */proc* file system. In GMA, a producer is a process that sends events to a directory service or consumers. A producer may also accept search queries from its local users or applications. Several systems have implemented their own producers, such as MDS GRIS, Ganglia monitor daemon (*gmond*), and Hawkeye Agent.

The key different between P-GMA and ordinary GMA is on the design of the registry or directory service. The GMA assumes a centralized registry or a hierarchically organized directory service like LDAP. In contrary, P-GMA leverages the recent research efforts on P2P-based indexing techniques to index and search resource information in a scalable P2P network. The indexing layer of P-GMA can be implemented by using various alternate schemes, such as MAAN[6], NodeWiz[4], or SWORD[14]. Applications in the consumer layer can directly search resources or monitor their status by issuing multi-attribute range queries to any nodes in the P2P indexing network. To monitor the global resource status, P-GMA builds an aggregation layer on top of the indexing layer with distributed aggregation trees. We will discuss the details of building a balanced aggregation tree in the rest of this paper. The consumer layer of P-GMA includes various essential applications for Grids, such as application scheduling, system diagnostics and capacity planning.

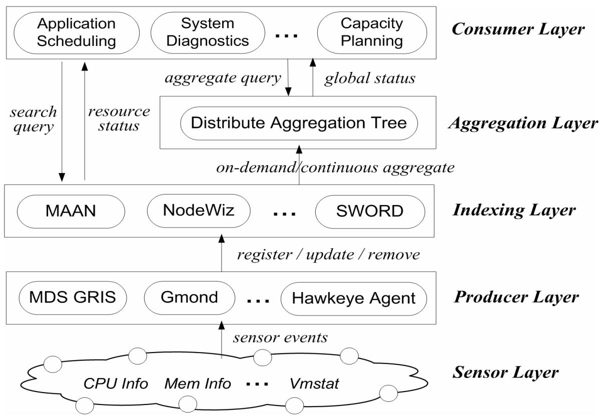


Figure 1: The architecture of P2P-based Grid resource monitoring

2.2. Multi-Attributed Addressable Network

We have proposed a *multi-attribute addressable network* called MAAN[6] to index Grid resources in a structured P2P network like Chord. In MAAN, a Grid resource is represented with a list of attribute-value pairs, such as (<*cpu-speed*, 2.8GHz>, <*memory-size*, 1GB>, <*cpu-usage*, 95%>, ...). MAAN stores each Grid resource on the Chord successor nodes of its attribute values. Suppose a resource has m pairs $<a_i, v_i>$ and $H_i(v)$ is the hash function for attribute a_i. Each resource will be stored at node $n_i = successor(H(v_i))$ for each attribute value v_i, where $1 \leq i \leq m$. A registration message for attribute value v_i is routed to its successor node using the Chord successor routing algorithm[18]. Thus, the routing hops for resource registration is $O(m \log n)$ for a resource with m attributes in a network of n nodes. Since numeric attribute values in MAAN are mapped to the Chord identifier space by using a locality preserving hash function H, numerically close values for the same attribute are stored on nearby nodes. Given a range query $[l, u]$ where l and u are the lower bound and upper bound respectively, nodes that contain attribute value $v \in [l, u]$ must have an identifier equal to or larger than *successor*($H(l)$) and equal to or less than successor($H(u)$).

Suppose a node wants to search for a resource with attribute value $v \in [l, u]$ for attribute a. It first uses the Chord routing algorithm to route it to node n_l, the successor of $H(l)$. Node n_l then finds its locally matched resources, and forwards the query to its successor if it is not the successor of $H(u)$, denoted by n_u. Otherwise, node n_u sends back the query result to the query originator. There are total $O(\log n + k)$ routing hops to resolve a range query for one attribute, where k is the number of nodes between n_l and n_u. Multi-attribute range queries are resolved by using a single-attribute dominated approach that only does 1-iteration around the Chord identifier space. It takes $O(\log n + n \times s_{min})$ routing hops to resolve the query, where s_{min} is the minimum selectivity of all sub-queries.

2.3. Distributed Aggregation Tree

In P-GMA, the aggregation problem can be formulated as follows. Consider a network of n nodes, each node i holds a local value $x_i(t) \in X$ in time slot t, where $1 \le i \le n$. For a given aggregate function $f: X^+ \rightarrow X$, the goal is to compute the aggregated value $g(t)$ of all local values in time, i.e. $g(t) = f(x_1(t), x_2(t), ..., x_n(t))$ in a decentralized fashion. To solve the above aggregation problem, we propose a *distributed aggregation tree* (DAT) approach that builds a tree structure implicitly from the native routing paths of Chord. In DAT, each node applies the given aggregate function f on the values of its child nodes, and sends the aggregated value to its parent node. By recursively aggregating the values through the tree in a bottom-up fashion, the root node will calculate the global aggregated value very efficiently since it only needs to collect the values from its direct children.

However, it is challenging to build aggregation trees *explicitly* by maintaining the parent-child membership [12]. First, explicit tree construction has limited scalability on a large number of aggregation trees since the parent-child maintenance overhead increases linearly with the number of trees. Second, the membership overhead will be further exaggerated when nodes dynamically join or leave the network. Instead of maintaining explicit parent-child membership, the DAT scheme uses the existing neighboring information of Chord to organize nodes into a tree structure in a bottom-up fashion. When a node joins or leaves the network, the Chord protocol will update its neighbors automatically using the finger stabilization algorithm [18]. Therefore, the DAT scheme does not have to repair the parent-child membership and significantly reduces the tree maintenance overhead.

In DAT, all nodes aggregate towards the global information with regard to a given object key called *rendezvous key*. A rendezvous key is the Chord identifier of a given aggregate index similar to the "*Group By*" clause in the SQL language. The rendezvous key is determined by DAT applications. For example, in Grid resource monitoring systems, the aggregated global resource attributes are indexed by different attribute names, e.g. *CPU usage*. In this case, the rendezvous key is the SHA1 hash value of the attribute name.

3. Load-Balancing DAT Algorithms

This section presents the design and analysis of two DAT construction algorithms based on different Chord routing schemes. The basic scheme builds a DAT tree from the finger routes of all Chord nodes to a given root node. To further balance the aggregation load among nodes, a new balanced routing scheme is proposed in Algorithm 1 to build more balanced DAT trees.

3.1. Structured P2P Network Model

We assume that the nodes will be self-organized into a Chord network [18]. We model the Chord network as an undirected graph G with n nodes. For a node v, let $\textsc{Id}(v)$ denote the unique identifier of v in a b-bit identifier space, where $\textsc{Id}(v) \in [0, 2^b)$. In Chord, the identifier space is structured as a cycle of 2^b, and the distance between two identifiers i_1 and i_2 is $\textsc{Dist}(i_1, i_2) = (i_1 + 2^b - i_2) \bmod 2^b$. Similar to [18], we use the term *node* to refer to both the node and its identifier. Chord assigns objects to nodes using a *consistent hashing* scheme. For an object stored in Chord, let k be its key in the same identifier space as nodes, i.e. $k \in [0, 2^b)$. Key k is assigned to the first node whose identifier is equal to or follows k in the circular space. This node is called the successor node of key k, denoted by *successor*(k).

All Chord nodes organize themselves into a ring topology according to their identifiers in the circular space. Besides its immediate predecessor and successor, each node also maintains a set of finger nodes that are spaced exponentially in the identifier space. The j-th finger of node v, denoted by $\textsc{Finger}(v, j)$, is the first node that succeeds v by at least 2^{j-1} in the identifier space, where $0 \le j < b$. A lookup message for key k is forwarded to its successor node by using the *finger routing* scheme. Let v be the successor node of k, and $f_{u,v}$ be the finger routing path (i.e. finger route) from u to v. Suppose $f_{u,v}$ is of the form $< w_0, w_1, ..., w_{q-1}, w_q >$, we have (1) $w_0=u$, $w_q=v$, and (2) for any $0 < i < q$, $w_{i+1} = \textsc{Finger}(w_i, j)$, such that $w_{i+1} \in (w_i, k]$ and $\textsc{Dist}(w_{i+1}, k) = \min\{\textsc{Dist}(\textsc{Finger}(w_i, j), k), 0 < j \le b\}$. Since the fingers of u are spaced exponentially in the identifier space, each hop in the finger route covers at least half of the identifier space (clockwise) between u and v.

3.2. Basic DAT Construction

The basic construction scheme builds a DAT tree on Chord in a bottom-up fashion. Let k be the rendezvous key of a given aggregation, and r be the root node of the DAT tree for this aggregation. The successor node of k is automatically selected as the root node via the same consistent hashing scheme as Chord, i.e. r=*successor*(k). Since consistent hashing has the advantage of mapping keys to nodes uniformly, this root selection scheme is capable of building multiple DAT trees in a load-balanced fashion. Besides the automatic selection of a root node, applications still have the flexibility of designating a given Chord node as the root by using its identifier as the rendezvous key.

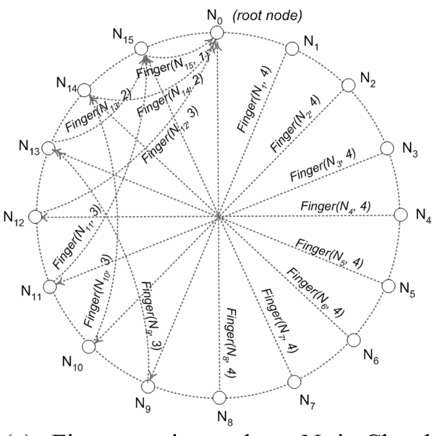
Considering a Chord network of n nodes, F is the set of finger routes from all nodes to a given root node r. We have $F=\{f_{v,r} | 1 \leq v \leq n\}$, where $f_{v,r}$ is the finger route from v to r as we specified in Sec. 3.2. To build a tree rooted at node r, each node uses the next hop of its finger route towards key k as its parent node. Intuitively, all finger routes destined to k will *implicitly* build a DAT tree, called *Basic DAT*. Let $p(v,i)$ be the next hop of v in $f_{i,r}$ from i to r, assuming $p(v,i)$ is empty if v is not in $f_{i,r}$ or v is the last hop of $f_{i,r}$. We have: (1) for any Chord finger route $f_{v,r}= < w_0,w_1,...,w_q >$ from node v to r, we have $w_i \neq w_j$ where $i \neq j$ and $0 \leq i, j \leq q$; (2) for any node $v \neq r$, the next hop of v in any finger route $f_{i,r}$ is the same, where $v \in f_{i,r}$ and $1 \leq i \leq n$.

Thus, each finger route is loop-free and each node except r has a unique parent node. Each node v has the same next hop $p(v,i)$ towards r regardless of finger route $f_{i,r}$. We can simply use $p(v,i)$ as the parent node of v to build a basic DAT tree $T(r)$. It is quite obvious that this scheme will construct a DAT tree rooted at r since each finger route is loop-free and each node except r has a unique parent node. Figure 2 illustrates an example of constructing a basic DAT rooted at node N_0 in a Chord network of 16 nodes with 4-bit identifiers. In Fig. 2(a), the label on each link, denoted by FINGER(N_i, j), represents that the j-th finger node of N_i is selected as the next hop of N_i towards the root node N_0. In this example, each finger route towards N_0 from a Chord node N_i corresponds to the path from N_i to the root in the basic DAT. For example, the finger route from N_1 to N_0 is $< N_1,N_9, N_{13}, N_{15}, N_0 >$ in Fig. 2(b), and the basic DAT has the same path from N_1 to N_0 as shown in Fig.2(b). Since N_0 is the next hop of N_8, N_{12}, N_{14}, and N_{15}, it has four child nodes correspondingly.

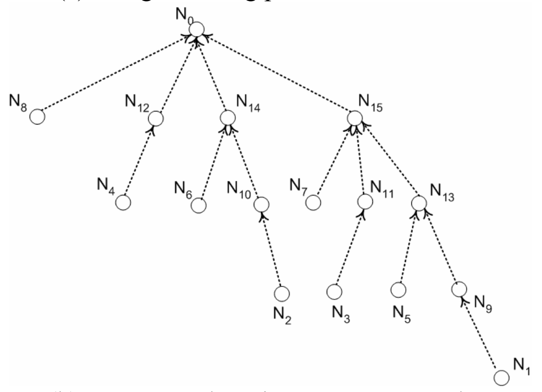
This basic DAT construction algorithm can be easily extended to a distributed setting. Actually distributed nodes do not need to build DAT trees explicitly. Instead, all the nodes know its parent directly by using the Chord finger routing; i.e., the next hop in the forwarding route is the parent. Since Chord has a very nice stabilization algorithm to update its fingers during node arrival and departure, the resultant DAT tree will adapt to node dynamics accordingly. Next, we will analysis two important properties of basic DAT: namely the *tree height* and *branching factor.*

3.3. Analysis of Basic DAT Properties

The height and branching factor of a DAT tree are important for the scalability and load-balance of distributed aggregation. The tree height determines the maximal number of nodes an aggregation message must traverse before reaching the root. The branching factor of a node is the number of children of the node. Since each node in basic DAT is responsible for aggregating the information from its children, its branching factor indicates the aggregation load of the node.



(a) Finger routing paths to N_0 in Chord


N_0
N_8 N_{12} N_{14} N_{15}
N_4 N_6 N_{10} N_7 N_{11} N_{13}
N_2 N_3 N_5 N_9
N_1

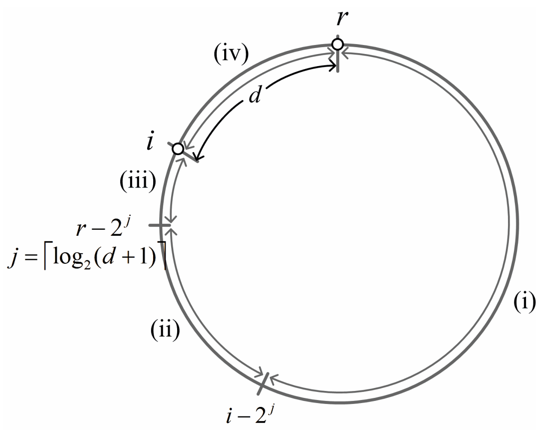
(b) Constructed Basic DAT tree rooted at N_0

Figure 2: Basic DAT tree construction using Chord finger routes to N_0 in a 16-node overlay.

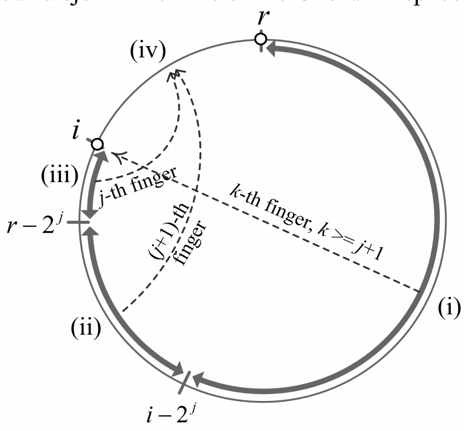
.

In basic DAT, the tree height of is O(log n) for a network of n nodes. This is because the tree height is equal to the length of the longest Chord finger route, which is $O(\log n)$ hops in a network of n nodes. From the example in Fig. 2(b), we know that the branching factor of a node is related to the distance between the node and the root. Let FINGER$^+(i, j)$ denote the j-th *outbound finger* of node i, we have FINGER$^+(i, j) = i + 2^{j-1} (\text{mod } 2^b)$, where j=1,2,...,b. Symmetrically, if v=FINGER$^+(i, j)$, we define i as the j-th *inbound finger* of v, denoted by FINGER$^-(v, j)$. Therefore, we have FINGER$^-(v, j)=v-2^{j-1}(\text{mod } 2^b)$. In the following proof, we assume that all arithmetic operations on Chord node identifiers are modulo operations of 2^b.

For a given node i, let PARENT(i) be the outbound finger of i that most closely precedes r. The children of i must be a subset of its inbound fingers. Not all inbound fingers of i will choose i as their parents since they may have other outbound fingers that are more close to r. Suppose node r is the root node, and $B(i, n)$ is the *branching factor* of node i in a basic DAT with n nodes. We consider $n=2^b$ and with index $i=0,1,2,...,2^b-1$. As shown in Fig. 3(a), the identifier space is divided into four disjoint intervals: (i) $(r, i-2^j]$, (ii) $(i-2^j, r-2^j]$, (iii) $(r-2^j, i)$, and (iv) $[i, r]$, where $j=\lceil \log_2 (d+1) \rceil$. Fig. 3(b) identifies the parents of nodes in interval (i), (ii), and (iii). Consider a basic DAT tree in which n *n*odes are evenly distributed in identifier space, the branching factor of node i is computed as follows: $B(i, n) = \log_2 n - \lceil \log_2 (d / d_0 + 1) \rceil$, where d=DIST(i, r) and d_0 is the distance between any two adjacent nodes.



(a) Four disjoint intervals of the Chord ID space



(b) Parent fingers of nodes in (i), (ii), and (iii)

Figure 3: Illustration of the parent fingers of nodes in different identifier spaces

The rigorous mathematical proof of this theorem is quite involved, details are given in our technical report [10]. We sketch the proof in two cases: (1) $1 < d < 2^{b-1}$, and (2) $2^{b-1} \leq d < 2^b$. For case (2), $B(i, n) = \log_2(n) - \lceil \log_2(d+1) \rceil = 0$. For case (1), the children of i are its inbound fingers in $(r, i-2^j]$, where $j = \lceil \log_2 (d+1) \rceil$. Thus, for case (2), node i has $B(i, n) = \log_2 n - j = \log_2 n - \lceil \log_2(d+1) \rceil$ children. When $n < 2^b$, we shrink the key space by a factor of $d_0=n/2^b$ to yield $B(i,n)=\log_2(n) - \lceil \log_2(d/d_0 + 1) \rceil$. Thus, The branching factor of a basic DAT is not the same for all nodes. For example, the root node has the maximal branching factor of $\log_2(n)$. However, the minimal branching factor of non-leaf nodes is 1 for nodes in the interval of $[r-nd_0/4, r-nd_0/2)$. Thus, the basic DAT is not balanced and some nodes need to aggregate information from many more child nodes than others. This prompts us to build more balance DAT trees.

3.4. Balanced DAT Construction

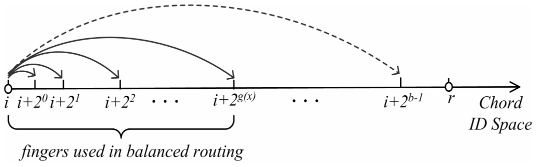
The imbalance of the basic DATs is due to the greedy strategy applied in the Chord finger routing algorithm. A Chord node always forwards a message to the closest preceding node in its finger table. For example, the node N_8 in Fig.2 forwards its update to the node N_0 directly, using the finger 2^3 away in the identifier space from itself. To build a *balanced* DAT with a constant number of branches, we propose a *balanced routing* scheme to construct the routing paths from all nodes to a given root node.

Instead of selecting a parent finger from the entire finger table, node i only considers a subset of fingers that are at most $2^{g(x)}$ away from i, where $g(x)$ is a function of the clockwise distance x between i and the root r in the identifier space. We call $g(x)$ the *finger limiting function* of node i. In Fig. 4, the solid arrows represent the fingers that could be used as a parent finger of i in the balanced routing scheme. The dashed arrow represents the parent finger that otherwise would be used by the ordinary finger routing scheme.

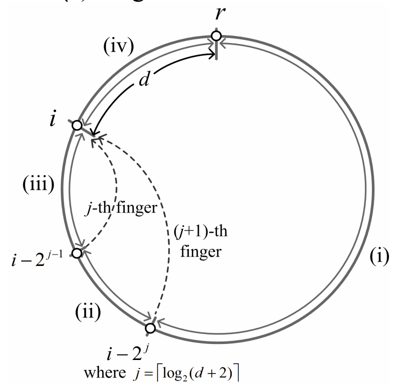
Next, we derive a function $g(x)$ such that all balanced routing paths (i.e. *balanced routes*) to r will build a balanced DAT tree with a constant branching factor, given nodes are evenly distributed in the identifier space. Intuitively, any given node i should have at most two contiguous inbound fingers that will use i as their parent fingers to r. For the ease of exposition, we will also assume that $n=2^b$ and $i=0,1,2,...,2^b-1$. Let d=DIST(i, r), and $j=\lceil \log_2(d+2) \rceil$. Suppose node u and v are the j-th and j+1-th inbound fingers of i respectively. The whole space can be divided into four disjoint intervals: (i) $(r, i-2^j)$, (ii) $[i-2^j, i-2^{j-1}]$, (iii) $(i-2^{j-1}, i]$, and (iv) $(i, r]$ as shown in Fig. 4(b).

To have a constant branching factor for each node, we will let u and v be the only two child nodes of a given node i. Therefore, the inbound fingers of i in interval (i) and (iii) must not use i as their next hop to r. For node v, we have

$$\begin{cases} x = r - (i - 2^j) = d + 2^{\lceil \log_2(d+2) \rceil} \\ g(x) = j = \lceil \log_2(d+2) \rceil \end{cases} \tag{1}$$



(a) Finger subset intervals



(b) Parent fingers of nodes in 4 intervals

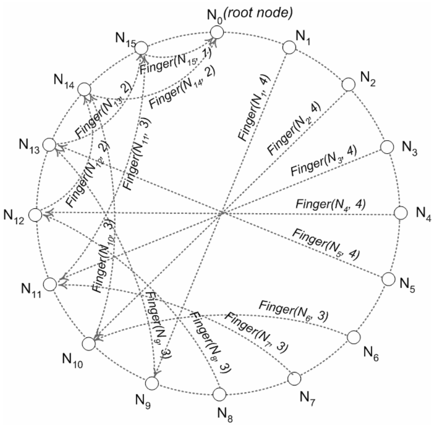
Figure 4: Subset of fingers used in balanced Chord routing scheme

Solving the above equation, we have $g(x)=\lceil \log_2((x+2)/3)\rceil$. In Sec. 4.2, we will show that each node has at most two children, i.e. the j-th and j+1-th inbound fingers. When $n < 2^b$, we shrink the identifier space by a factor of $d_0=n/2^b$ since nodes are evenly distributed. Therefore, $g(x)=\lceil \log_2((x+2d_0)/3)\rceil$, where d_0 is the distance between two adjacent nodes. Algorithm 2 specifies the construction of a balanced DAT.

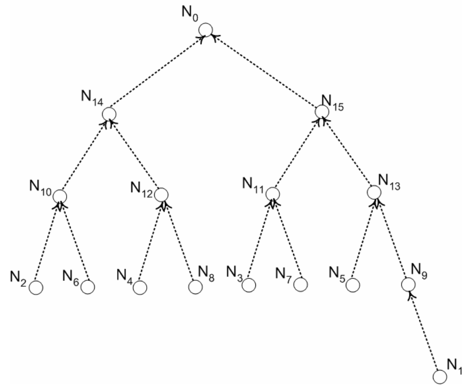
Algorithm 1 Balanced DAT Construction

```
1:  INPUT: rendezvous key $k$, finger table FINGER($i, j$)
    of each node $i$, where $i$=1,2,...,n, and
           $j$=0,1,...,b-1.
2:  OUTPUT: a balanced DAT tree $T$ rooted at node
    $r$=successor($k$)
3:  $d_0 \leftarrow$ average distance between two adjacent nodes
4:  for $i \leftarrow 1$ to $n$ do
5:      if DIST($k, i$) < DIST(PRED($i$), $i$) then
6:          ROOT(T) ← $i$
7:      endif
8:      $x \leftarrow$ DIST($i, k$)
9:      max ← $\lceil \log_2((x+2d_0)/3)\rceil$
10:     for $j \leftarrow$max downto 0 do
11:         if DIST($i$, FINGER($i, j$)) ≤ DIST($i, k$) then
12:             PARENT($i$) ← FINGER($i, j$)
13:         endif
14:     endfor
15: endfor
```

Figure 5(a, b) demonstrate this balanced routing scheme and the almost balanced DAT tree. In Fig. 5(a), node N_8 only selects the closest preceding finger from the fingers that are at most 2^2 hops away from itself, since x=0−8 mod 2^4=8, and $g(x) = \lceil \log_2(8 + 2)/3\rceil = 2$. Therefore, node N_1 now is the next hop of N_8, while node N_0 was its next hop in Fig. 2(a) when the ordinary finger routing algorithm was used. The routing of all other nodes remain unchanged and the balanced DAT tree is balanced with a maximum branching factor of 2 as shown in Fig. 5(b).



(a) Balanced routing paths to N_0



(b) Balanced DAT tree rooted at N_0

Figure 5: Building a balanced DAT trees by using the balanced routing scheme

3.5. Analysis of Balanced DAT Properties

We now analyze the branching factor and tree height of balanced DAT. When all nodes are evenly distributed in the identifier space, we show that the resulting DAT from balanced routing is indeed a well balanced tree with maximum branching factor of 2. Consider a balanced DAT tree with evenly distributed node identifiers, its tree height is at most $\log_2(n)$ for n nodes. As shown in Fig. 4(b), node u is the closest child to i and DIST(u, i) = 2^{j-1}. We prove DIST(u,i)≥ d in the following two cases: (a) $d=2^k$, and (b) $d=2^k-1$, where k=0,1,...,2^{b-1}. When $d=2^k$, DIST(u,i) = $2^{\lceil \log_2(d+2)\rceil-1} = 2^k = d$. Similarly, when $d = 2^k-1$, DIST(u,i) = $2^{\lceil \log_2(d+2)\rceil-1} = 2^k = d+1 > d$. Since the distance between i

and its child is at least the same as the distance between i and r, the length of any balanced routing path is at most $\log_2(n)$ in a network of n nodes. Therefore, the tree height of balanced DAT is at most $\log_2(n)$ as well.

For any given node i, only its j-th and j+1-th inbound finger in (r,i) are the children of i in a balanced DAT. We discuss the following four cases:

(1) $\forall w \in (r, i-2^j)$, we have $i \neq \text{PARENT}(w)$ since $w+2^j < i$;

(2) $\forall w \in (i-2^{j-1}, i)$, we have $i \neq \text{PARENT}(w)$ since $w+2^{j-1} \in (i,r)$;

(3) $w=i-2^{j-1}$, we have $i = \text{PARENT}(w)$ since $g(d+2^{j-1}) = \left\lceil \log_2((d+2^{\lceil \log_2(d+2) \rceil -1}+2)/3) \right\rceil = \lceil \log_2(d+2) \rceil - 1 = j\text{-}1$;

(4) $w=i-2^j$, we have $i=\text{PARENT}(w)$ since $g(d+2^j) = \left\lceil \log_2((d+2^{\lceil \log_2(d+2) \rceil}+2)/3) \right\rceil = \lceil \log_2(d+2) \rceil = j$.

In addition, a DAT must be a balanced tree if its tree height is $\log_2(n)$ and the branching factor is at most 2. For any given node i, its left sub-tree should have at most one more node than its right sub-tree, and vice versa. Otherwise, the overall tree height will be more than $\log_2(n)$ for a tree of n nodes since the branching factor is at most 2.

Thus, if the ranges between two immediately adjacent nodes are the same, the balanced routing scheme will lead to a balanced DAT tree. However, if the interval of a randomly selected node is split as that in Chord, the ranges will not be uniformly distributed [1]. The ratio of the maximal and minimal ranges is $O(\log n)$, where n is the network size. To ensure the ranges among nodes distributed uniformly, Adler et al[1] proposed an *identifier probing* approach in which each joining node probes $O(\log n)$ neighbors of a randomly selected node and splits the one with the maximal interval. The ratio of the maximal and minimal ranges in this approach is bounded by a constant factor. Our simulation results in Sec. 6.2 show that with node identifier probing, the maximal branches in the balanced DAT will be a constant as well.

4. DAT Prototype Implementation

Based on the above DAT construction algorithms, we implemented a prototype system of DAT, called *libdat*, in C language on both Linux and FreeBSD. Next, we will describe the architecture of our DAT implementation, and detailed mechanisms on identifier probing and aggregation synchronization. Fig. 6 shows the implementation architecture of our DAT prototype. In this implementation, each DAT node consists of three layers, i.e. RPC, Chord and DAT layers. The RPC layer implements the low-level mechanisms of remote procedure call for the communication among distributed nodes. A *RPC manager* module is implemented ar the socket-level to send and receive UDP packets. To simplify the testing and evaluation of our DAT prototype, we also implemented a discrete event simulation engine that provides the same interface to the Chord and DAT layers. A heap-based event queue is used to insert and fire those events in a chronological order. Without modifying the upper layers, the simulator can be used to evaluate the performance of *libdat* with large number of nodes as we show in Sec. 5.

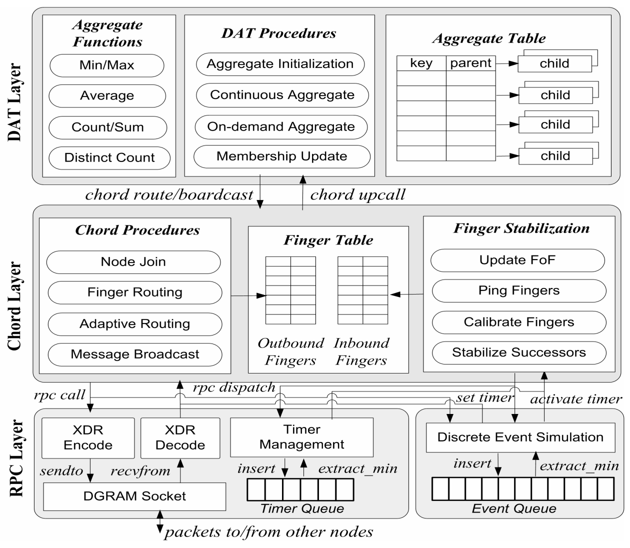


Figure 6: DAT Implementation Architecture

The Chord layer extends the original Chord protocols with extensions on identifier probing and maintaining extra information about fingers. It consists of three components, i.e. Chord procedures, finger table and finger stabilization. Each node keeps not only the information of its direct fingers, but also the information of its *fingers of finger* (FOF). When a node joins the network, it first sends a join request with a random identifier to a well-known node. Then the request is forwarded to the successor of the random identifier. The successor splits the maximal interval of its fingers and returns the designated node identifier to the joining node. Finally the node uses the same node join operation as in Chord [18] to join the network.

The DAT layer implements both on-demand and continuous aggregate modes for different aggregation functions. It leverages the three underlying Chord routines, i.e. route, broadcast and upcall. To support multiple DAT trees simultaneously, each DAT node also maintains an aggregation table that keeps track of the current active DAT trees as shown in Fig. 6. When a node initializes an aggregate for a given rendezvous key, it adds a new entry in the aggregation table for this aggregate, and computes its child nodes based on the information in the Chord finger table.

5. Experimental Results

In this section, we measure the performance and scalability of our DAT prototype system with three metrics, including tree properties, message overhead, and effects of load balancing.

5.1. Experiment Setup

To faithfully evaluate the DAT system at different scales, we have implemented a UDP-based RPC module as well as a discrete event simulator. We deployed the DAT system in an 8-node cluster at the USC Internet and Grid Computing Lab. The cluster nodes are dual Xeon 3.0 GHz processors with 2 gigabytes of memory running Linux kernel 2.6.9 and connected via a 1-Gigabit Ethernet switch. We ran up to 64 DAT instances on each machine to create a network of 512 nodes. For larger networks up to 8192 nodes, we ran the DAT prototype in the event-driven simulator. Note that both RPC-based and simulator-based setups use the same Chord and DAT layers. They indeed have the consistent results for the metrics we measured in this section.

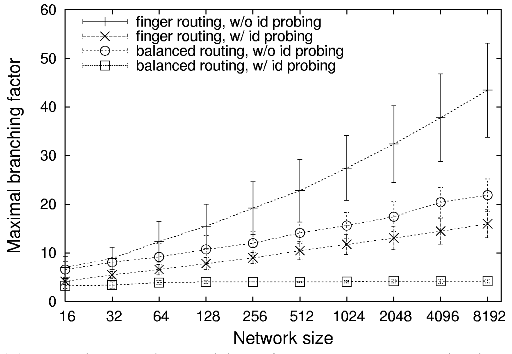
5.2. Measured DAT Tree Properties

We examine the DAT tree properties with various network sizes from 16 to 8192. We studied three different properties of DAT trees, i.e. maximum and average branching factors. Fig. 5(a) shows the maximal branching factor as a function of network size for both basic and balanced DATs. The maximal branching factor of the basic DAT increases on a log scale with the number of nodes. Note that the network size is in log scale. When probing is used to balance node identifiers, the maximal branching factor decreases significantly, e.g. 16 vs. 43 for 8192 nodes. However, it still increases on a log scale with network size. In contrast, the maximal branching factor of balanced DAT is almost a constant of 4 when node identifiers are uniformly distributed by probing $O(\log n)$ neighbors. However, without identifier probing, balanced DAT trees still have the maximal branching factor that increases on a log scale. This is due to the ratio of the maximal and minimal ranges between adjacent nodes is $O(\log n)$ when node identifies are randomly chosen.

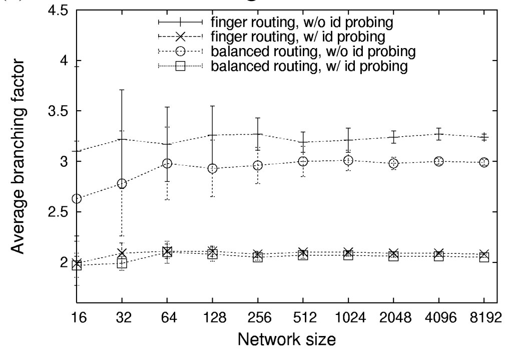
Figure 5(b) shows that the average branching factors of balanced DAT are constant as the network size increases. When identifier probing is used, two DAT trees have almost the same constant average branching factor of 2. However, they increase to 3 and 3.2 respectively if there is no identifier probing, although they remain constant as network size increases.

5.3. Effects of Load Balancing

Besides the average message overhead per node, the distribution of aggregation messages among nodes is another important metric to evaluate the performance of the DAT system. Apparently, the evener the messages are distributed among nodes, the better the aggregation process is load balanced. Fig. 9(a) plots the distributions of aggregation message in a network of 512 nodes for three different schemes. In this figure, the DAT nodes are sorted in the descending order of the number of aggregation messages. We define *node rank* as the position of a node in this sorted node list. As shown in Fig. 8(a), the message distribution of the centralized scheme without DAT is quite skewed. Note that the y-axis is in a log-scale. For example, the root node is the most loaded one with 511 aggregation messages, which is almost the same as the total number of nodes in the network. This is because each node in the network except the root node itself must send their local values to the root node directly. In addition, the closer a node precedes the root node in the Chord identifier space, the more aggregation messages it has to forward for other nodes due to the nature of the Chord finger routing algorithm.



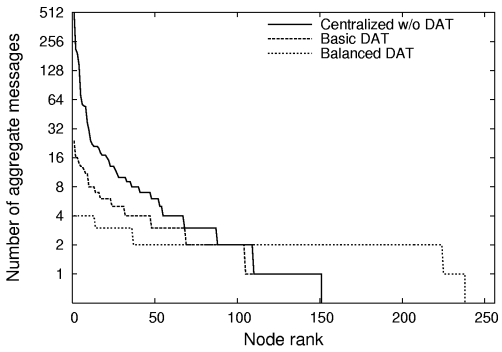
(a) Maximum branching factor vs. network size.



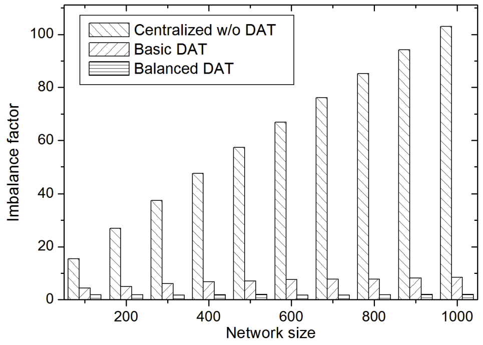
(b) Average branching factor vs. network size

Figure 7: Comparison of tree properties for different DAT schemes

In contrast, distributed aggregation in the network with DAT trees significantly reduces the imbalanced load at the root monitor. Each intermediate node in the DAT tree only processes the aggregation messages from its direct children instead of every node in the sub-tree. For example, the most loaded nodes in basic and balanced DATs have only 24 and 4 messages respectively. Since basic DAT is not a balanced aggregation tree, the root has more children than other nodes. Therefore, the distribution of message overhead in basic DAT is still more skewed than that in balanced DAT.



(a) Distribution of aggregation messages among nodes



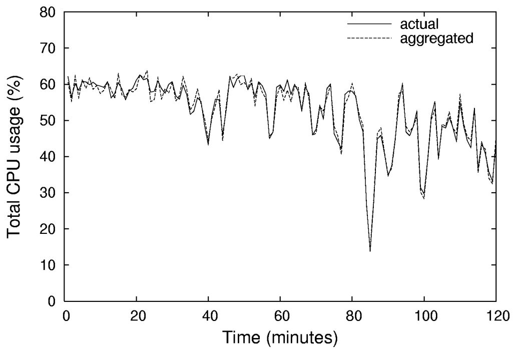
(b) Imbalance of aggregation messages vs. network size

Figure 8: Comparison of load balance for centralized, basic and balanced DAT schemes

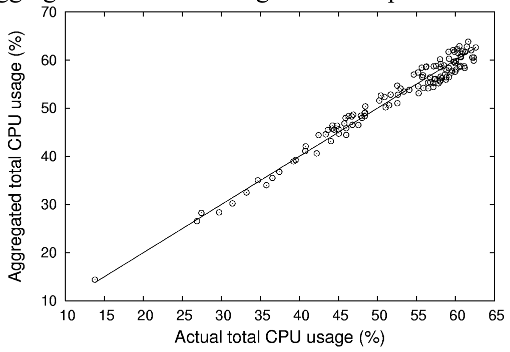
We define the *imbalance factor* of message overhead as the ratio between the maximum and average number of aggregation messages on each node. The aggregation is well balanced if the imbalance factor is close to 1. Fig. 8(b) shows the imbalance factor as a function of the network size varying from 100 to 1000 for three difference aggregation schemes. The imbalance factor of the centralized scheme increases almost linearly with the network size since the root node has to process O(n) aggregation messages. The imbalance factor of the basic DAT only increases on a log-scale with the network size. For example, the imbalance factors are 4.2 and 8.5 for the networks of 100 and 1000 nodes respectively. The balanced DAT has an almost constant imbalance factor under different network sizes, e.g. 1.9 and 2.0 for 100 and 1000 nodes respectively. This further validates our theoretical analysis of the DAT tree properties in Sec. 4.2 and Sec. 4.4.

5.4. Accuracy of Grid Resource Monitoring

Figure 9 illustrates an example of aggregating the global average CPU usage in a simulated Grid with 512 nodes. We collected a 2-hour long trace of the CPU usages on an 8-processor Sun Fire v880 server at USC. We then simulated a Grid with 512 nodes, and each node has the same CPU usage as in the trace. Fig. 9(a) plots the total CPU usages over the time period of 2 hours. The solid and dotted lines show the actual and aggregated usages respectively. Fig. 9(b) plots the actual vs. aggregated CPU usages where the solid line shows the equality. As most points are clustered around the diagonal, our DAT scheme achieves a very accurate aggregation of the global CPU usages.



(a) Aggregated total CPU usage in a time period of 2 hours



(b) Aggregated vs. actual total CPU usage

Figure 9: Aggregated CPU usage in 2 hours for a simulated Grid with 512 nodes

6. Related Work

Many Grid resource monitoring and discovery systems [3][4][6][8][9][13][14][15] are related to our research. Zanikolas and Skellariou[22] has surveyed these systems in a scope-oriented taxonomy with great details. Due to the limited space, we will not discuss them in this paper.

Our work on DAT is related to several previous research efforts on aggregating the global information in distributed systems. Astrolabe [16] provides a DNS-like distributed management service by grouping nodes into non-overlapping zones and specifying a tree structure of zones. Several aggregation schemes have been proposed to leverage the topology information of structured P2P networks [2][12][17][24]. SOMO[24] offers an information gathering and disseminating infrastructure on top of arbitrary DHTs. The SOMO tree is built by recursively dividing the DHT identifier space into disjoint regions and assigning each region to a DHT node. DASIS[2] and Willow[17] use a similar scheme to build a single aggregation tree on hypercube-based DHTs, such as Pastry. By aggregating the depth information, DASIS improves the node joining algorithm for better load balance[2].

Li et al [12] build an aggregation tree by mapping nodes to their parents in the tree with a parent function. By adjusting parameters in a parent function, their approach can build multiple interior-node-disjoint trees to tolerate single points of failure. SDIMS[21] is the most closely related project to our work. In SDIMS, each attribute is hashed on to a key and corresponding aggregation tree is built from Plaxton routes to the key. The aggregation trees in SDIMS are similar to the basic DATs built from Chord finger routes. Our work focuses more on the construction algorithms of more balanced aggregation trees.

7. Conclusions

We have presented the DAT algorithms, prototype implementation, performance evaluation and application on P2P-based Grid resource monitoring. Our work extends previous methods for distributed information aggregation. Summarized below are four major contributions: (1) We proposed a P2P-based architecture for scalable Grid resource monitoring and discovery; (2) A balanced DAT scheme is developed on Chord overlays to aggregate the global information in an efficient and load-balanced fashion. (3) The prototype DAT has been successfully evaluated with good tree properties, message overheads, and load balancing. (4) We demonstrate that the DAT scheme performs well in Grid resource monitoring and other applications.

For continuing efforts, we suggest to investigate the performance of DAT under extreme node dynamics. For example, it would be meaningful to test the DAT prototype system through benchmark experiments in a wide-area environments such as the PlanetLab or the DETER testbed. With the introduction of scalable aggregation schemes, many killer applications are now enabled to explore distributed resources in P2P and Grid computing systems.

References

[1] M. Adler, E. Halperin, R. M. Karp, and V. V. Vazirani, "A Stochastic Process on the Hypercube With Applications to Peer-to-Peer Networks," *Proc. of the 35th* STOC, June 2003.

[2] K. Albrecht, R. Arnold, M. Gahwiler, and R. Wattenhofer, "Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave," *Proc. of the 4th Int'l Conf. on Peer-to-Peer Computing*, 2004.

[3] M.A. Baker, G.C. Smith, "GridRM: An Extensible Resource Monitoring System", Proc. of the IEEE International Cluster Computing Conference, 1–4 December 2003, pp. 207–214.

[4] S. Basu, S. Banerjee, P. Sharma, and S.-J. Lee, " NodeWiz: Peer-to-Peer Resource Discovery for Grids", *Proc. of Cluster Computing and the Grid (CCGrid)*, 2005,

[5] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating System Support for Planetary-Scale Network Services", *Proc. of the 1st Symp. on Networked Systems Design and Implementation (NSDI)*, 2004.

[6] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A Mulit-Attribute Addressable Network for Grid Information Services," *Journal of Grid Computing*, no. 1, pp. 3-14, 2004.

[7] M. Cai, K. Hwang, "Distributed Aggregation Schemes for Scalable Peer-to-Peer and Grid Computing", submitted to *IEEE Trans. on Parallel and Distributed Systems*, Sept,2006.

[8] A. W. Cooke et al., "The Relational Grid Monitoring Architecture: Mediating Information about the Grid", *Journal of Grid Computing*, vol. 2, no. 4, December 2004.

[9] S. Czajkowski, K. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing", *Proc. of HPDC*, 2001..

[10] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *The Int'l Journal of Supercomputer Applications and High Performance Computing*, 11(2), 1997.

[11] E. Korpela, D. Werthimer, D. Anderson, J. Cobb and M. Lebofsky. "SETI@Home - Massively Distributed Computing for SETI", *Computing in Science & Engineering*, Jan. 2001.

[12] J. Li, K. Sollins, and D.-Y. Lim, "Implementing Aggregation and Broadcast over Distributed Hash Tables," *SIGCOMM Computer and Communication Review*, vol. 35, no. 1, 2005.

[13] M.L. Massie, B.N. Chun, D.E. Culler, "Ganglia Distributed Monitoring System: Design, Implementation, and Experience", *Parallel Computing*, vol. 30, 2004, pp. 817–840.

[14] D. Oppenheimer, J. Albrecht, D. Patterson and A. Vahdat. "Design and Implementation Tradeoffs for Wide-Area Resource Discovery". *Proc. of HPDC*, July 2005.

[15] KyoungSoo Park and Vivek S. Pai, "CoMon: A Mostly-Scalable Monitoring System for Planetlab", *Operating Systems Review*, Vol 40, No 1, Jan 2006.

[16] R. V. Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining," *ACM Transaction on Computer Systems*, 21(2), pp. 164-206, 2003.

[17] R. V. Renesse and A. Bozdog, "Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol," *Proc. of the Int'l Workshop on Peer-to-Peer Systems* (IPTPS '04), Feb. 2004.

[18] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc.of SIGCOMM*, 2001.

[19] P. Thibodeau, "Planet-Scale grid," *ComputerWorld*, October 10, 2005.

[20] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V.Taylor, R. Wolski, "A Grid Monitoring Architecture", *GWDPerf-16–3, Global Grid Forum*, August 2002.

[21] P. Yalagandula and M. Dahlin, "A Scalable Distributed Information Management System," *SIGCOMM*, 2004.

[22] S. Zanikolas, R. Sakellariou, "A Taxonomy of Grid Monitoring Systems", *Future Generation Computer Systems*, vol. 21, 2005, pp. 163-188.

[23] X. Zhang, J. Freschl, and J. M. Schopf, "A Performance Study of Monitoring and Information Services for Distributed Systems," Proc. of HPDC, 2003.

[24] Z. Zhang, S.-M. Shi, and J. Zhu, "SOMO: Self-organized Metadata Overlay for Resource Management in P2P DHT," in *Proc. of the Int'l Workshop on Peer-to-Peer Systems*, 2003.