Distributed Aggregation Schemes for Scalable Peer-to-Peer and Grid Computing[*]

Min Cai, *Student Member IEEE* and **Kai Hwang**, *Fellow IEEE*

Abstract:

Peer-to-Peer (P2P) systems and P2P Grids are evolving as two viable distributed computing paradigms for wide-area resource sharing on the Internet. To achieve scalable distribution of processing workload among the nodes, large-scale distributed systems such as P2P Grids need to acquire some global information in a fully decentralized fashion. This paper presents efficient schemes for building *distributed aggregation trees* (DAT) on a structured P2P network like Chord. By leveraging the topology and routing mechanisms of Chord, 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 proves 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, P2P reputation management, distributed RDF repository, and worm signature generation systems.

Index Terms: ***Peer-to-Peer systems, P2P Grids, distributed aggregation, load balancing, distributed hash table (DHT), Chord system, Grid resource monitoring, RDF repository, P2P reputation systems,*** and ***performance evaluation.***

[*] Manuscript submitted to *IEEE Transactions on Parallel and Distributed Systems* (TPDS) on Sept. 18, 2006. This research was supported in part by NSF ITR grant ACI-0325409. M. Cai and K. Hwang are with the University of Southern California, Los Angeles, CA 90089, {mincai, kaihwang}@usc.edu. Corresponding author: Kai Hwang, Email: kaihwang@usc.edu or Tel. (213) 740-4470 or Fax: 213 740-4418.

1 Introduction

With pervasive use of computers and high-speed networks, *Peer-to-Peer* (P2P) systems [19][22][27][28] and P2P Grids [8][29] are emerging as two viable approaches to sharing digital contents and computing resources over the Internet. In contrast to the traditional *client-server* architecture, the P2P paradigm employs a flat and symmetric structure to distribute computations, storage and communication among the nodes for better scalability, load balance, and failure tolerance. Large-scale P2P file-sharing networks, e.g Gnutella, KazaA and eDonkey, construct an unstructured overlay network of millions of nodes to store and retrieve files in a fully decentralized fashion. *Distributed Hash Tables* (DHT), e.g. CAN [23], Chord [28], Tapestry [33], and Pastry [27], offer a scalable key lookup service by organizing nodes into a given topology (e.g. ring or hypercube) and routing messages within logarithmic number of hops. P2P Grid such as the SETI @Home [14] achieves massively distributed computing by aggregating a large number of CPU cycles from millions of contributing client computers [29].

The emphasis of existing P2P and Grid systems is on the scalable distribution of processing workload among nodes without any central coordination. On the other hand, large-scale distributed systems in general and P2P systems in particular, need to acquire some global information of the whole system. Examples of global system properties include network size and total free storage in P2P file-sharing networks, and total CPU usages in P2P Grids. Moreover, distributed intrusion detection systems have to continuously monitor some global measurement metrics from several sites. For example, a worm signature can be detected by counting its global fingerprint repetition and distinct IP addresses [5].

Distributed aggregation is an essential building block for computing the global information in large-scale distributed systems. An *aggregate function*, e.g. *min*, *max*, *count*, and *sum*, takes a set of input values, and calculates a single output value that summarizes the inputs. In a distributed environment, there are two alternatives, i.e. *centralized* and *distributed*, to yield a global value by aggregating local values from all nodes. The former collects all local values at a single node and aggregates them directly using an aggregate function. The latter recursively applies the aggregation function on a subset of local values until the global value is generated.

Aggregating towards the global information posts a major challenge to P2P systems due to their large-scale and decentralized nature. Centralized aggregation does not scale well to a large number of nodes and has the drawback of a single point of failure. For this purpose, P2P systems deliberatively eliminate any centralized server that interacts with all nodes. Besides, distributed aggregation requires the coordination among nodes so that the aggregation can be done gradually in the network. A *distributed aggregation tree* (DAT) is necessary for aggregating local values recursively. In DAT, each node applies the same aggregate function on local values of its children. The root node yield the global value aggregated from all nodes. However, it is non-trivial to maintain a distributed tree structure in dynamic P2P systems.

Any effective scheme for distributed aggregation in large-scale P2P systems has to meet the requirements on scalability, adaptiveness, and load balancing. First, the scalability has two criteria. To scale to a large number of nodes, each aggregation should only introduce a limited number of messages with respect to the network size. To scale to a large number of DAT trees, the scheme should have low construction and maintenance overhead for each tree. Second, the aggregation scheme has to adapt to the dynamics of node arrival and departure. The node insertion and deletion in DAT should have minimal impact to the aggregation process. 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.

To meet the above challenges, we propose a distributed aggregation scheme that provides a common interface for aggregating global information. Our scheme constructs a DAT tree among nodes by leveraging the structured P2P network, i.e. Chord [23]. 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 with up to 8192 nodes. We study DAT applications on Grid resource monitoring [7][8], P2P reputation management [35], distributed RDF repository [6], and distributed worm signature generation[5].

The remainder of this paper is organized as follows: Sec. 2 reviews the related work. Section 3 models the aggregation problem and introduces the DAT system. We present the DAT construction algorithms in Sec. 4 and a prototype implementation in Sec. 5. Sections 6 and 7 report performance results on three real-life DAT applications. Finally, we conclude with a summary of contributions and suggest further research work.

2 Related Work

Our work on DAT is related to several previous research efforts on aggregating the global information in distributed systems [10][16][24]. Astrolabe [24] provides a DNS-like distributed management service by grouping nodes into non-overlapping zones and specifying a tree structure of zones. In Astrolabe, a representative node is elected for each zone to propagate information across zones using a gossip protocol. Bawa et al [3] compared the tree-based and propagation-based schemes for estimating aggregates in P2P networks. They showed that static aggregation trees are often prone to node failures in unstructured P2P networks.

Most structured P2P networks dynamically adjust their topologies during node joining and leaving. Several aggregation schemes have been proposed to leverage the topology information of structured P2P networks [2][9][15][25][32]. SOMO[32] 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[25] use a similar scheme to build a single aggregation tree on hypercube-based DHTs, such as Pastry[27], Tapestry[33], and Kademlia [19]. By aggregating the depth information, DASIS improves the node joining algorithm for better load balance[2].

Li et al [15] 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[31] 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 default Plaxton routing algorithm is also modified to provide administrative isolation. The aggregation trees in SDIMS are similar to the DATs built from Chord finger routes. Our work focuses more on the construction algorithms of more balanced aggregation trees.

Instead of building aggregation trees, gossip-based protocols[4][11][13] estimate the aggregates by exchanging information among nodes in an epidemic manner. Kempe et al [13] show that these protocols converge exponentially fast to the true aggregates. In addition to P2P networks, in-network information aggregation has been investigated in the context of sensor networks with more restrict energy and security constraints [20][30][34].

Table 1: Five Applications of Distributed Information Aggregation

Applications	Information	Index	Functions	Mode	Systems
Grid resource monitoring	Resource attributes	Attribute name	Sum, Average	Continuous	Globus MDS[8], MAAN[7]
P2P reputation aggregation	Top-*k* reputable peers	Peer name	Top-*k*	Continuous	EigenTrust[12], PowerTrust[35]
P2P RDF repository	Object values	Predicate name	Sum, count, etc	On-demand	Edutella[21], RDFPeers[6]
Distributed worm signature generation	Fingerprint statistics	Fingerprint	Sum, distinct-count	Continuous	WormShield[5]
Wide-area network monitoring	IP flow statistics	Flow attribute name	Sum, average	On-demand	MIND[17]

Distributed information aggregation has a broad application on various distributed systems, such as Grid resource monitoring in MDS[8] and MAAN[7], P2P reputation aggregation in EigenTrust[12] and PowerTrust[35], and so on. Table 2 summarizes five example applications of distributed information aggregation in different contexts. These applications are compared in terms of four important aspects: *aggregated information, aggregate index, aggregate function, and aggregate mode.* For example, the aggregated information of Grid resource monitoring systems is the global properties of Grid resources, such as total CPU usages.

In the case of Grid resource monitoring, the global properties of Grid resources are aggregated as in Globus/MDS and MANN. The aggregated information is indexed by *aggregation index* that is similar to the "*Group By"* clause in the SQL language. Different aggregate functions are used to calculate the global information with two aggregation modes, i.e. on-demand and continuous. The former calculates the global information once upon an aggregation request, while the latter does the aggregation continuously for every time period. In this paper, we will report the benchmark results on three example DAT applications in Grid resource monitoring, P2P reputation aggregation, and P2P RDF repository. More results on distributed worm signature generation can be found in our earlier report[5], which will not be repeated here.

3 DAT System Model

In this section, we formulate the distributed aggregation problem in a structured P2P network. We then describe the basic formulation of the DAT approach based on Chord[28].

3.1 The Distributed Aggregation Problem

The distributed aggregation problem can be formulated as follows. Consider a P2P network modeled as a undirected graph $G=(V,E)$, where the vertex set V contains n nodes and E is the set of links between nodes. In the network, each node i holds a local value $x_i(t) \in X$ in time slot t, where $1 \leq i \leq n$. 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.

We are interested in any generic aggregate function that is known as reduction function. A aggregate function f is said to be *reducible,* if $f(X_1 \cup X_2) = f(f(X_1), f(X_2))$, where $X_1, X_2 \subset X$ and $X_1 \cap X_2 = \phi$. As shown in [9], most aggregate functions are *reducible,* such as *count*, *min*, *max*, *sum*, *average*, and *distinct_count*. The reducible aggregation functions have two important properties. First, the aggregates either return a single representative value from the set of all values (e.g. *min* and *max*), or calculate some property of all values (e.g. *count*, and *sum*). In both cases, the output value has much smaller size than the set of input values. Second, the reduction aggregate functions are applied to a large set of input values recursively, i.e. a subset of input values at each time.

3.2 Structured P2P Network Model

P2P networks can be classified into two categories in terms of their topology structures, i.e. unstructured and structured networks [18][36]. In this paper, we assume that the nodes in our aggregation problem will be self-organized into a structured P2P network, such as CAN[23], Chord[28], Tapestry[33], and Pastry[27]. Particularly, we use the Chord network proposed by Stoica et al [28] as the underlying infrastructure for distributed aggregation.

We model the Chord network as an undirected graph $G=(V,E)$ with $n=|V|$ nodes, and the links are the overlay connections between nodes. For node $v \in V$, let $\mathrm{ID}(v)$ denote the unique identifier of v in a b-bit identifier space, where $\mathrm{ID}(v) \in [0, 2^b)$. 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 $\mathrm{DIST}(i_1, i_2) = (i_1 + 2^b - i_2) \bmod 2^b$. Similar to [28], we use the term *node* to refer to both the node and its identifier.

Consistent Hashing: 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).

Network Topology: All Chord nodes organize themselves into a ring topology according to their identifiers in the circular space. For node v, let $\mathrm{PRED}(v)$ denote its immediate predecessor, and

SUCC(v) denote its immediate successor. Each Chord 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 FINGER(v, j), is the first node that succeeds v by at least 2^{j-1} in the identifier space, where $0 \leq j < b$. Therefore, the finger table contains more nearby nodes than faraway nodes at a doubling distance.

Finger Routing: A lookup message for key k is forwarded to its successor node by using the *finger routing* scheme in Chord. When a node u wants to lookup k that is far away from u, it forwards the lookup message to the finger node whose identifier most immediately precedes the successor node of k. By repeating this process, the message gets closer and closer to, and will eventually reach the successor node. 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} = FINGER(w_i,j), such that $w_{i+1} \in (w_i, k]$ and DIST(w_{i+1},k) = min{ DIST(FINGER(w_i, j), k), $0 < j \leq b$ }.

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.

3.3 Basic Formulation of the DAT Model

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 child nodes. Fig. 1 shows an example of aggregating the global value through a DAT tree of seven nodes, where each node n_i has a local value x_i and i=1,2,...,7. After applying the aggregate function three times at nodes N_5, N_6, and N_7, the root node N_7 will calculate the global aggregated value from all local values.

In a dynamic P2P network with node joining and leaving, it is quite challenging to build aggregation trees *explicitly* by maintaining the parent-child membership [15]. 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. Since a node is often part of every aggregation tree, all related trees have to be adjusted upon its departure.

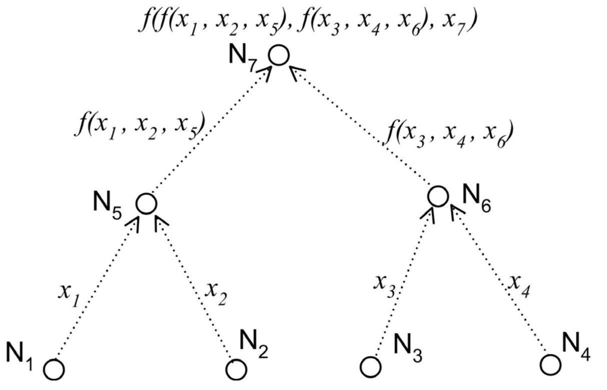


Figure 1: Example of aggregating the global value through a 7-node DAT tree

As shown in Sec. 3.2, structured P2P networks, e.g. Chord, have a unique network topology and routing mechanism that provide an elegant infrastructure for building aggregation trees implicitly. Instead of maintaining explicit parent-child membership, DAT nodes use the existing neighboring information of Chord to organize themselves 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 [28]. Therefore, the DAT scheme does not have to repair the parent-child membership and significantly reduces the tree maintenance overhead. Next, we describe the detailed DAT construction algorithms.

4 DAT Construction Algorithms

This section presents the design and analysis of two DAT construction algorithms based on different Chord routing schemes. The basic Algorithm 1 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 2 to build more balanced DAT trees.

4.1 Basic DAT Construction

The basic construction scheme builds a DAT tree on Chord in a bottom-up fashion. We assume that 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 properties are indexed by different property names, e.g. *CPU usage*. In this case, the rendezvous key is the SHA1 hash value of the property name.

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. For example, in the WormShield system [5], a DAT tree has to be built for each fingerprint. By using this scheme, each monitor will be responsible for aggregating the information of roughly the same number of fingerprints. 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 node 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*. The following two lemmas prove this conjecture since each finger route is loop-free and each node except r has a unique parent node.

Lemma 1 For any Chord finger route $f_{v,r} = < w_0, w_1, \ldots, w_q >$ from node v to r, we have $w_i \neq w_j$ where $i \neq j$ and $0 \leq i, j \leq q$.

Proof: According to the Chord finger routing algorithm discussed in Sec. 3.2, the next hop of a node towards key k is its finger node that most immediately precedes *successor*(k). In our context, r=*successor*(k). Hence, we have $\text{DIST}(w_i, r) < \text{DIST}(w_j, r)$, where $i > j$ and $0 \leq i,j \leq q$, and $0 < \text{DIST}(w_i, r) < \text{DIST}(w_0, r)$, where $0 < i < q$. Since the last hop of route $f_{v,r}$ is r, i.e. $w_q=r$, node r is guaranteed to be reached within one circle of the ring topology. This implies that a node will appear at most once in any given finger route to node r. **Q.E.D.**

Lemma 1 shows that each finger route destined to r is loop-free. Next, we will prove that a node has the same next hop in all finger routes that contains this node. 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 $f_{i,r}$.

Lemma 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 $1 \leq i \leq n$.

Proof: For a given key k, the next hop of v in $f_{i,r}$ is $p(v,i)$= FINGER(v,j) such that $p(v,i) \in (v, k]$ and DIST($p(v,i)$, k) = min{DIST(FINGER(v,j), k), $0 < j \leq b$}. Therefore, if $v \neq r$, there is at least one finger of v that is the next hop towards k. Moreover, the next hop is only determined by v and k, which is independent of the previous hops along the route. Hence, for any $v \neq r$, there is one and only one next hop of v in any finger route $f_{i,r}$, where $v \in f_{i,r}$ and $1 \leq i \leq n$. **Q.E.D.**

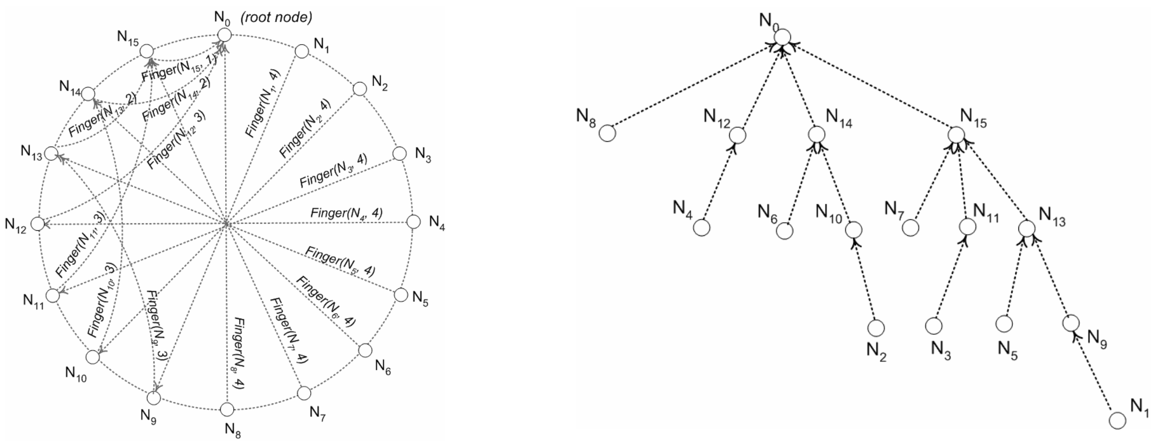
Since each node v has the same next hop $p(v,i)$ towards r regardless of finger route $f_{i,r}$, we can simply use $p(v,i)$ as the parent node of v to build a basic DAT tree $T(r)$ as specified in Algorithm 1. From Lemma 1 and 2, it is quite obvious that Algorithm 1 will construct a DAT tree rooted at r since each finger route is loop-free and each node except r has a unique parent node.

Algorithm 1 Basic DAT Construction Algorithm

```
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 basic DAT tree T rooted at node r=successor(k)
3:  for i←1 to n do
4:     if DIST(k, i) < DIST(PRED(i), i) then
5:        ROOT(T) ← i
6:     endif
7:     for j←b-1  downto 0 do
8:        if DIST(i, FINGER(i, j)) ≤ DIST(i, k) then
9:           PARENT(i) ← FINGER(i, j)
10:       endif
11:    endfor
12: endfor
```

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*.



(a) Finger routing paths to N_0 in Chord **(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.

4.2 Analysis of Basic DAT Properties

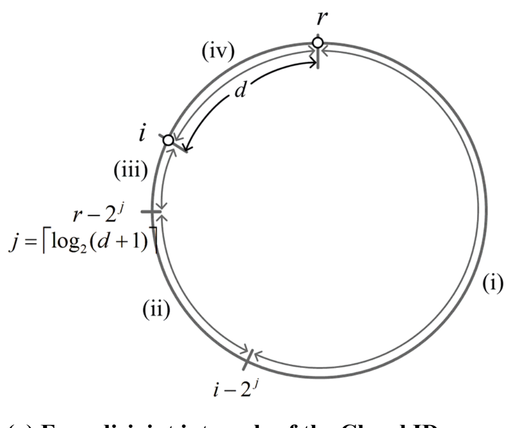
The height and branching factor of a DAT tree are important for the scalability and load-balance of distributed aggregation. The tree height determines the maximal number of nodes an aggregation message must traverse before reaching the root. Apparently, the aggregation latency increases as the DAT tree becomes deeper. 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. To avoid hotspots and balance the load among nodes, it is desired for each node to have the same branching factor. We formally analyze two tree properties of basic DAT in the following lemma and theorem.

Lemma 3 The tree height of basic DAT is O(log n) for a network of n nodes.

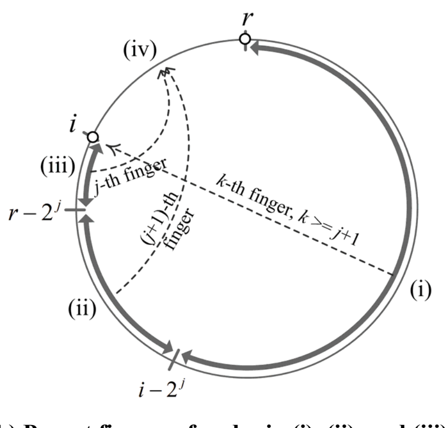
Proof: The basic DAT tree height is equal to the length of the longest Chord finger route, which is $O(\log n)$ hops in a network of n nodes. **Q.E.D.**

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 $\text{F}\small{\text{INGER}}^{+}(i, j)$ denote the j-th *outbound finger* of node i, we have $\text{F}\small{\text{INGER}}^{+}(i, j) = i + 2^{j-1} (\text{mod } 2^b)$, where j=1,2,...,b. Symmetrically, if v=$\text{F}\small{\text{INGER}}^{+}(i, j)$, 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).



(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

Theorem 1: 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.

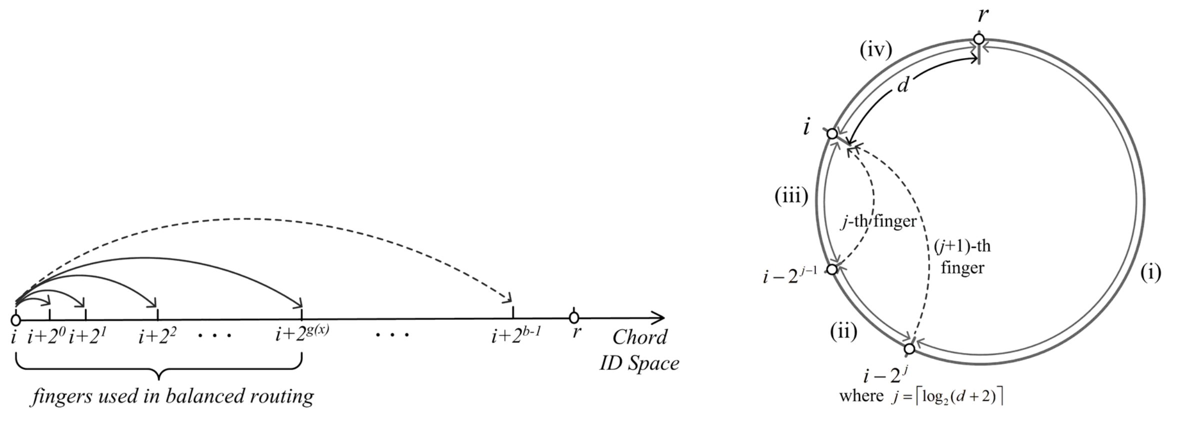
Proof : The rigorous mathematical proof of this theorem is quite involved, details are given in Appendix A. Only the proof sketch is outlined below: We prove two cases: (1) $1< d < 2^{b-1}$, and (2) $2^{b-1} \le d < 2^b$. For case (2), $B(i, n) = \log_2(n) - \lceil \log_2(d+1) \rceil = 0$. For case (1), 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 $B(i,n)=\log_2(n) - \lceil \log_2(d/d_0 + 1) \rceil$. **Q.E.D.**

Theorem 1 shows that 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.

4.3 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.



(a) Finger subset intervals **(b) Parent fingers of nodes in 4 intervals**

Figure 4: Subset of fingers used in balanced Chord 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 $d=\textsc{Dist}(i, r)$, 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

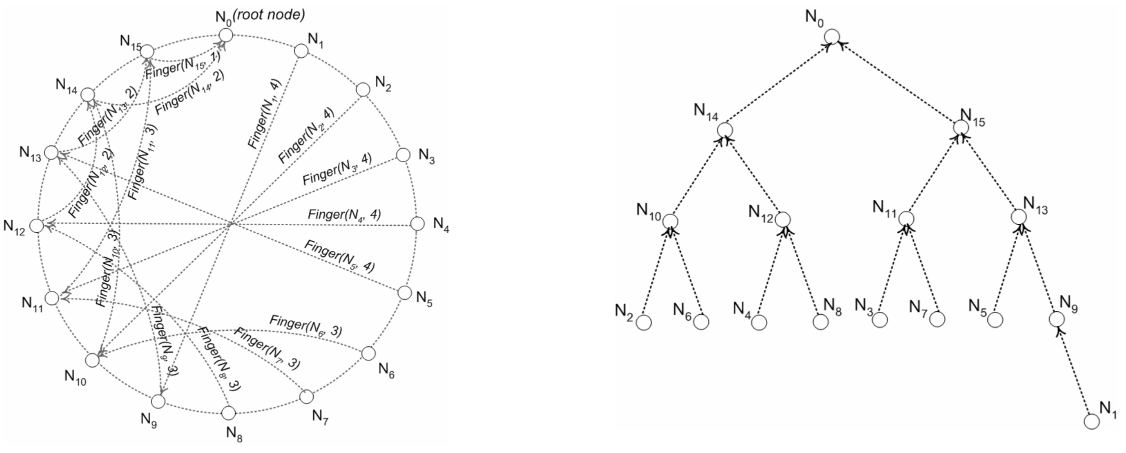
$$\begin{cases} x = r - (i - 2^j) = d + 2^{\lceil \log_2(d+2) \rceil} \\ g(x) = j = \lceil \log_2(d+2) \rceil \end{cases} \quad \textbf{(1)}$$

Solving the above equation, we have $g(x)=\lceil \log_2((x+2)/3) \rceil$. For detailed mathematical deduction, readers are referred to Appendix B. In Sec. 4.2, we will prove 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 2 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) $\leftarrow i$
7:     endif
8:     $x \leftarrow$ DIST($i, k$)
9:     max $\leftarrow \lceil \log_2((x+2d_0)/3) \rceil$
10:    for $j \leftarrow$ max  downto 0 do
11:       if DIST($i$, FINGER($i, j$)) $\le$ DIST($i, k$) then
12:          PARENT($i$) $\leftarrow$ 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 $x=0-8 \bmod 2^4=8$, 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 N0 **(b) Balanced DAT tree rooted at N0**

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

4.4 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, the following theorem proves that the resulting DAT from balanced routing is indeed a well balanced tree with maximum branching factor of 2.

Theorem 2: Consider a balanced DAT tree with evenly distributed node identifiers, its tree height is at most $\log_2(n)$ for n nodes.

Proof: As shown in Fig. 4(b), node u is the closest child to i and $\textsc{Dist}(u, i) = 2^{j-1}$. We prove $\textsc{Dist}(u,i) \geq d$ in the following two cases: (a) $d=2^k$, and (b) $d=2^k-1$, where $k=0,1,\ldots,2^{b-1}$. When $d=2^k$, $\textsc{Dist}(u,i) = 2^{\lceil \log_2(d+2) \rceil - 1} = 2^k = d$. When $d=2^k-1$, $\textsc{Dist}(u,i) = 2^{\lceil \log_2(d+2) \rceil - 1} = 2^k = d+1 > d$. 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. **Q.E.D.**

Theorem 3: Given nodes are evenly distributed in identifier space, the DAT tree built from balanced routes is a balanced tree with a branching factor of at most 2.

Proof: 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 split our discussion in four cases: (1) $\forall w \in (r, i-2^j)$, we have $i \neq \textsc{Parent}(w)$ since $w+2^j < i$; (2) $\forall w \in (i-2^{j-1}, i)$, we have $i \neq \textsc{Parent}(w)$ since $w+2^{j-1} \in (i,r)$; (3) $w=i-2^{j-1}$, we have i= PARENT(w) since $g(d+2^{j-1})=\left\lceil \log_2((d+2^{\lceil \log_2(d+2)\rceil -1}+2)/3)\right\rceil=\lceil \log_2(d+2)\rceil-1=j-1$; (4) w=i–2^j, we have i= 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. **Q.E.D.**

Theorem 3 proves that 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.

5 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.

5.1 Implementation Architecture

Figure 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. It provides four routines to its upper layers, i.e. *rpc_call*, *rpc_dispatch*, *set_timer*, and *activate_timer*. Both Chord and DAT messages are transmitted via UDP protocol in *external data representation* (XDR) format. The XDR encoding and decoding routines are automatically generated via the *rpcgen* compiler. A *RPC manager* module is implemented ar the socket-level to send and receive UDP packets. By using *select*() call that is waiting on the UDP socket, the *timer* *management* module is able to activate the next timer without being blocked.

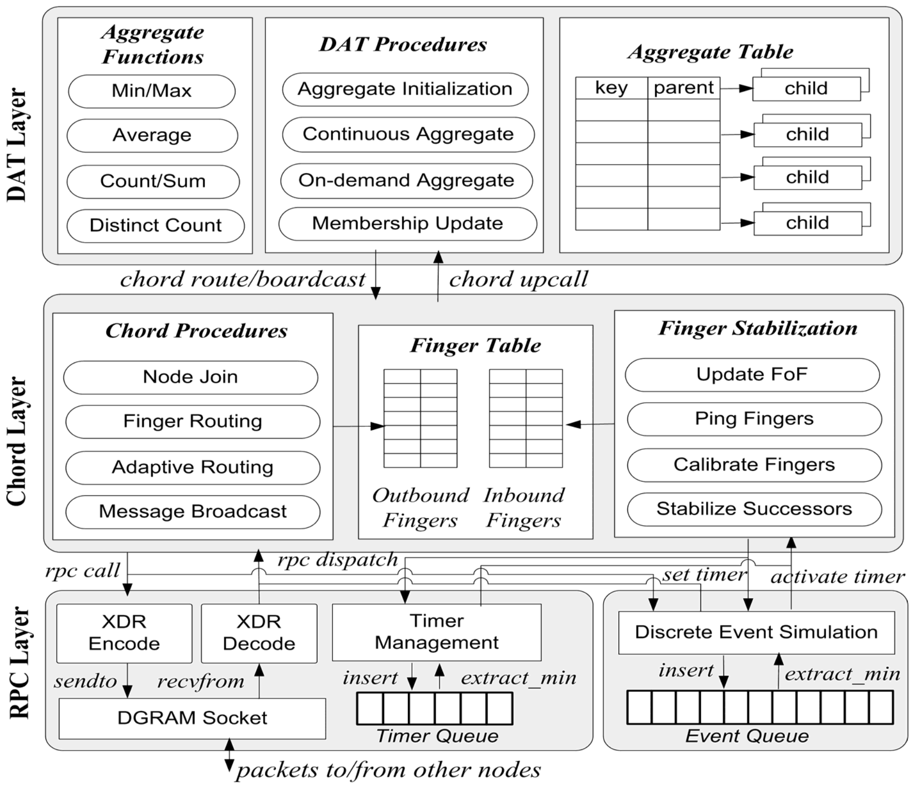


Figure 6: DAT Implementation Architecture

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. 6.

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 [28] 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. Next, we will describe the detailed mechanism of on-demand and continuous aggregation.

5.2 Aggregation Synchronization

During the course of aggregation, each intermediate node in the DAT tree must synchronize with its child nodes to compute the aggregated value of its sub-tree. The DAT system uses two different schemes for on-demand and continuous aggregations. For on-demand aggregation, each node only keeps the aggregate entry until it receives all values from its child nodes. With the information of fingers of finger, each DAT node can easily identify which inbound finger is its child for a given key *k*. During an aggregation, the root node first added an entry in the aggregate table, and sends a request to each child. After receiving the reported values from all children, it removes the aggregate entry, aggregates all values and returns the aggregated value to the client. This process is done recursively in top-down fashion at all nodes in the DAT tree.

In contrast, for continuous aggregation, the DAT system uses a bottom-up approach to aggregate the global value periodically. The root node first broadcast a aggregate request to all nodes in the network. Each node then computes its children information and added an entry in is aggregate table. In every aggregation period, each leaf node sends its local value to its parent node. Once receiving all updates from its child nodes or the time period expires, the parent node calculates its own aggregated value and send it to the parent node. The same process will be done recursively in a bottom-up fashion. Thus, the root node can aggregate the global value continuously for each time period.

6 Experimental Results on the DAT Prototype System

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.

6.1 Experiment Setup

To faithfully evaluate the DAT system at different scales, we have implemented a UDP-based

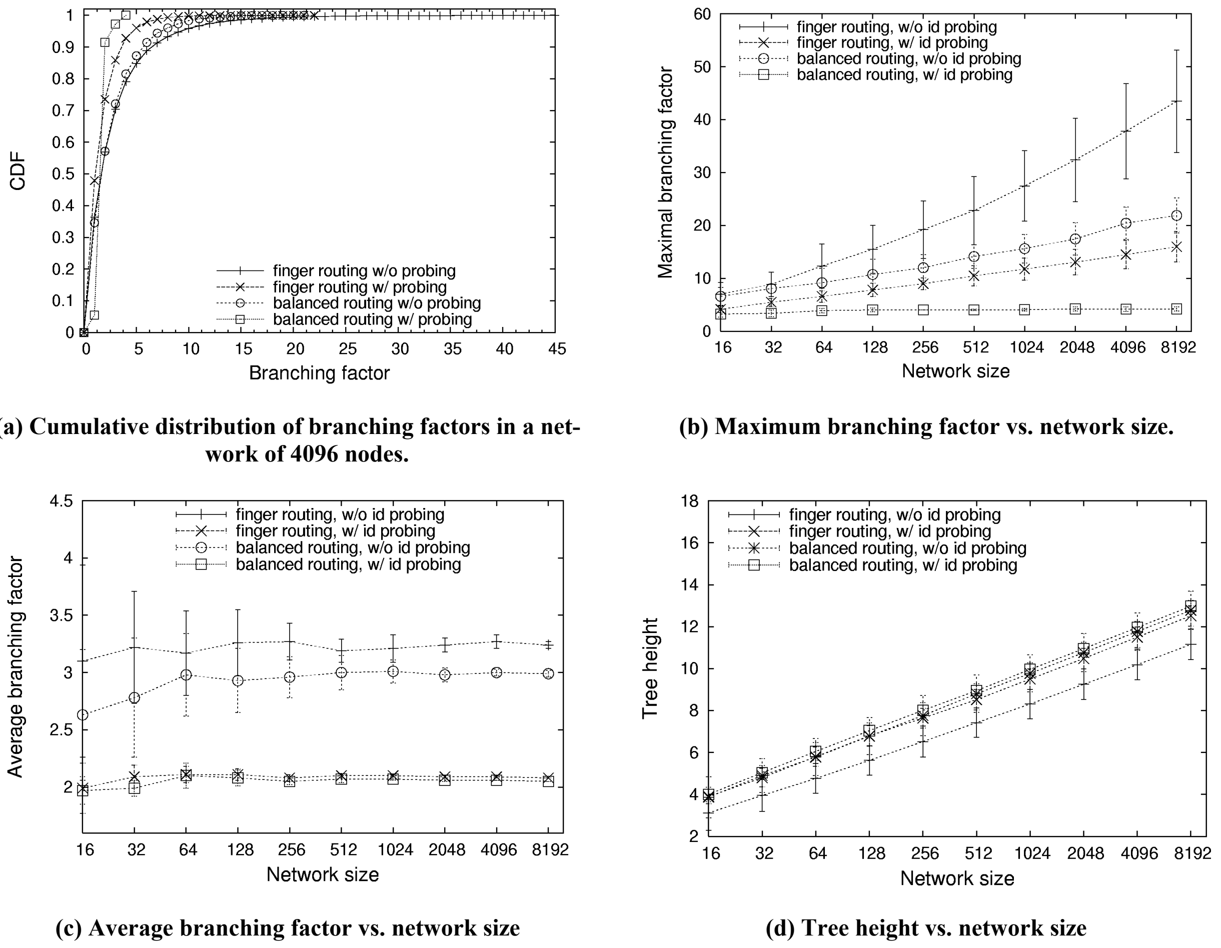
RPC module as well as a discrete event simulator. We deployed the DAT system in an 8-node cluster at the USC Internet and Grid Computing Lab. The cluster nodes are dual Xeon 3.0 GHz processors with 2 gigabytes of memory running Linux kernel 2.6.9 and connected via a 1-Gigabit Ethernet switch. We ran 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.

6.2 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 branching factor, average branching factor and tree height. Fig. 5(a) shows the cumulative distribution of DAT node branching factors in a network of 4096 nodes. When no probing scheme is used, both finger routing and balanced routing construct DAT trees with skewed distributions of branching factors. When both balanced routing and identifier probing are used, there are 5.5%, 86%, 5.6% nodes having branching factors of 1, 2, 3, respectively. And only about 3% remaining nodes have the maximal branching factor of 4. This implies that balanced routing paths with identifier probing construct much more balanced DAT trees than using other schemes.

Figure 5(b) plots 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(c) 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. As shown in Fig. 5(d), the tree height is always bounded by $O(\log n)$ since the routing hops are at most $O(\log n)$ in a network of n nodes.



(a) Cumulative distribution of branching factors in a network of 4096 nodes.

(b) Maximum branching factor vs. network size.

(c) Average branching factor vs. network size

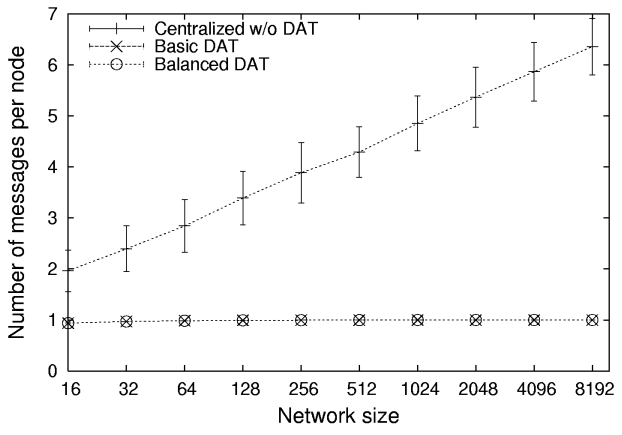
(d) Tree height vs. network size

Figure 7: Comparison of tree properties for different DAT schemes.

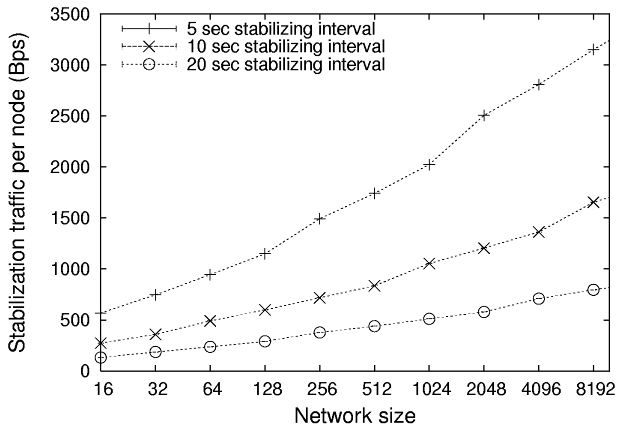
6.3 Measured Message Overheads

For distributed information aggregation systems, message overhead is an important metric to measure their scalability. There are two types of messages introduced by the DAT system, i.e. aggregation and stabilization messages. The former are transmitted among DAT nodes to aggregate the global information. Since the aggregation messages often vary by different workload, we measure their overhead by the number of messages used per node in each aggregation round. On the other hand, the stabilization messages are used by the Chord layer to keep the finger information up to date. Their overhead is measured by the stabilization traffic rate per node.

Figure 8(a) compares the aggregation message overhead under various network sizes for three aggregation schemes, i.e. centralized without DAT, basic DAT and balanced DAT. In the centralized scheme, each node updates its local value to the root node directly using the Chord finger routing. Therefore, the number of aggregation messages per node increases on a log-scale with network size since each update message will routed in $O(\log n)$ hops. For example, in a network of 1024 nodes, the centralized scheme uses 4.8 aggregation messages on average. In contrast, both basic and balanced DATs use one aggregation message per node since the local values of a sub-tree are aggregated and sent to its parent node only once.



(a) Number of aggregation messages for COUNT function vs. network size



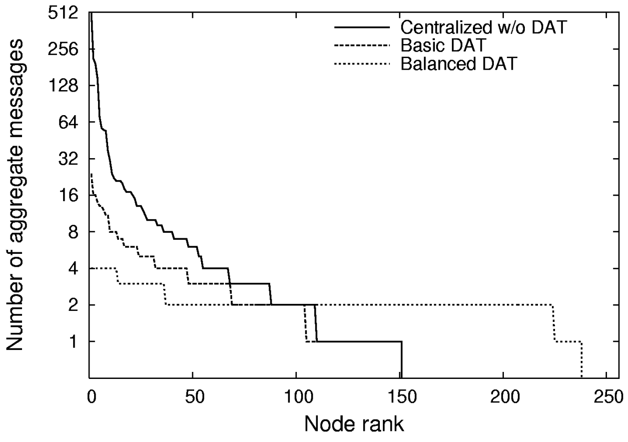
(b) Number of aggregation messages for AVERAGE function vs. network size

Figure 8: Aggregation overhead of centralized, basic and balanced DAT schemes with network size varying from 16 to 8192.

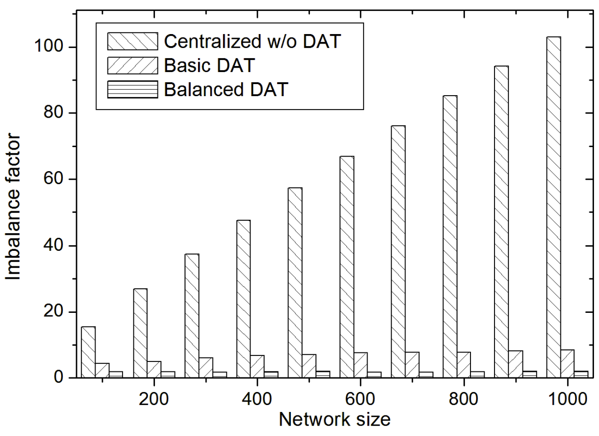
Next, we show the amount of stabilization overhead required to maintain the Chord overlay network. In the DAT system, each node periodically sends ping messages to its fingers to retrieve the information of its fingers of finger. It also sends Chord stabilization messages to their immediate successors; these messages ask nodes to identify their predecessor nodes. Finally, additional messages are sent periodically to maintain an updated finger table, in which each DAT node maintains pointers to nodes that are logarithmically distributed around the Chord identifier space. We refer collectively to these three types of messages for Chord membership maintenance as stabilization traffic. Fig 8(b) shows the measured overhead per node in bytes per second for stabilization traffic as the network size increases. The three lines show different periods (5s, 10s and 20s) at which the stabilization messages are sent. For all three update intervals, the stabilization traffic is quite low (less than 3 KB/s for 8192 nodes). The stabilization traffic per node increases at a rate of $O(\log^2 n)$ for a network of size n. This is because each node has O(log n) fingers and each ping message contains the information of O(log n) fingers of finger.

6.4 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.



(a) Distribution of aggregation messages among nodes



(b) Imbalance of aggregation messages vs. network size

Figure 9: Comparison of load balance for centralized, basic and balanced DAT schemes

As shown in Fig. 9(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.

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.

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. 9(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.

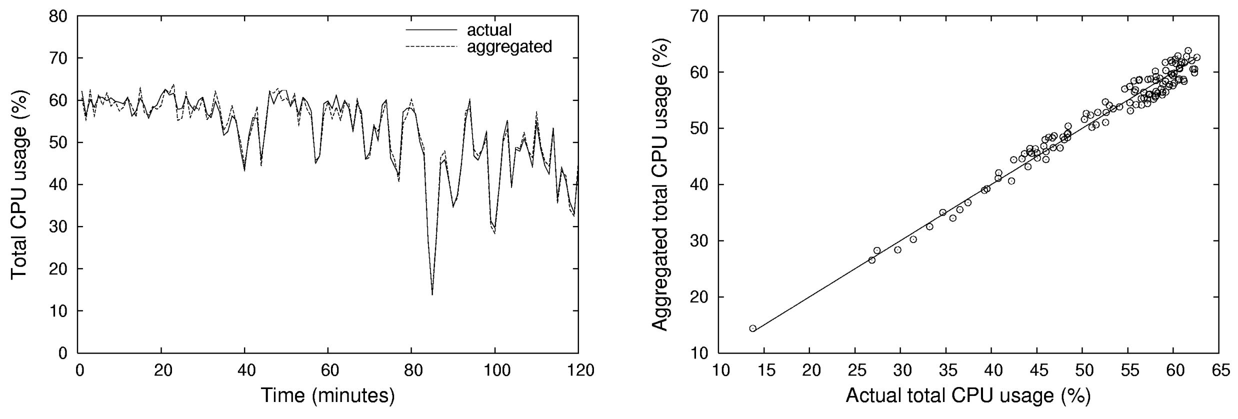
7 Scalable Applications in P2P Grids and P2P Systems

This section presents three real-world DAT applications in Grid resource monitoring [8], P2P reputation system[12][35], and P2P RDF repository[6][21]. We also briefly discuss other DAT applications in distributed worm containment[5] and wide-area network monitoring [17].

7.1 P2P Grid Resource Monitoring

Grid computing on a large scale requires scalable and efficient resource monitoring and discovery. We have developed a new structured P2P system, called *Multi-Attribute Addressable Network* (MAAN), to index and query Grid resources [7]. It extends traditional DHT systems to handle multi-attribute range queries. In MAAN, resources can be registered with a set of attribute-value pairs and can be searched by multi-attribute based range queries.

However, to monitor the global usages of resources, such as CPU, memory, and storage, we have to aggregate the global information from distributed nodes in the MAAN network. We extend the MAAN system with DAT trees to aggregate information in a scalable and load-balanced fashion. For a given resource attribute (e.g. CPU usage), a rendezvous key is automatically generated by hashing the attribute name with SHA-1. The global resource values are continuously aggregated with the specified aggregate functions, such as *sum* and *average*.



(a) Aggregated total CPU usage in a time period of 2 hours **(b) Aggregated vs. actual total CPU usage**

Figure 10: Aggregated CPU usage in 2 hours for a simulated P2P Grid with 512 nodes

Figure 10 illustrates an example of aggregating the global average CPU usage in a simulated P2P 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. 10(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. 10(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.

7.2 P2P Global Reputation Aggregation

P2P reputation systems are essential to evaluate the trustworthiness of participating peers and to combat the selfish and malicious peer behaviors. Inspired by the power-law distribution of user feedbacks on eBay, Zhou and Hwang have developed a PowerTrust system [35] that dynamically selects a few most reputable nodes, called power nodes. In PowerTrust, a distributed ranking scheme sorts the nodes in descending order of their global reputation scores and select the top-*k* most reputable nodes. This scheme requires $O(n \log n)$ messages for each ranking process.

Indeed, the problem of ranking top-*k* reputable nodes can be modeled as a distributed aggregation problem in which the aggregate function at each node selects top-*k* reputable nodes from its sub-tree. By applying this function recursively on a DAT tree, the root node will be able to select the top-*k* nodes for the whole P2P network. However, in a dynamic P2P network, if a node in the DAT tree fails, all the nodes in its sub-tree will not be selected in the top-*k* list. Particularly, if the root node fails, the whole ranking process will fail. Therefore, instead of using a single DAT tree, we can build multiple DAT trees with independent root nodes.

The top-*k* nodes will be selected from the results returned by all root nodes. Let ***A*** and ***B*** be the sets of real and ranked top-*k* nodes respectively. We define *ranking error* β as following: $\beta = |A - B|/|A|$. Fig. 11 plots the ranking error of top 16 reputable nodes out of 512 ones as a function of the number DAT trees for both 1% and 5% node fail-rates. As shown in Fig. 11, the ranking error drops sharply as the number of DAT trees increases for both 1% and 5% fail-rates.

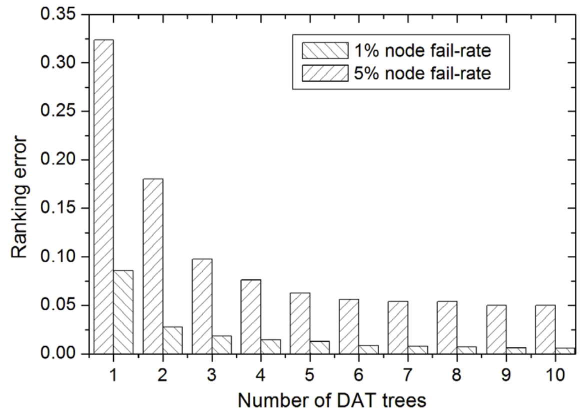


Figure 11: Ranking error of top 16 reputable nodes out of 512 ones drops sharply as DAT trees increase.

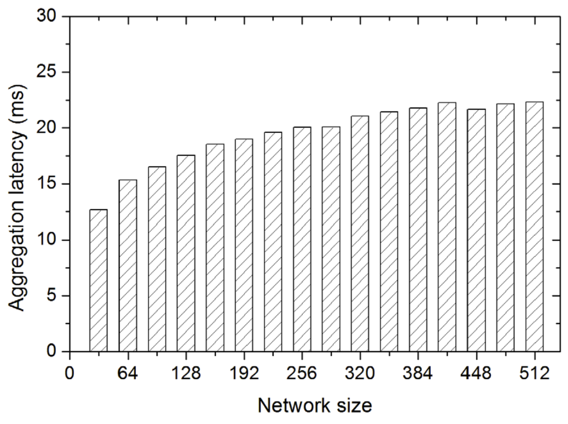


Figure 12: Aggregation latency in RDFPeers increases on a log-scale with the number of nodes.

For example, when only a single DAT tree is used to rank the top-16 nodes, the ranking errors are 0.32 and 0.08 for 1% and 5% fail-rates respectively. However, when two independent DAT trees are used, the ranking errors drop to 0.18 and 0.02 respectively. A real top-*k* node will be selected when it is in the top-*k* list of any DAT tree. Therefore, the probability of a real top-*k* node not being selected should decrease exponentially with the number of DAT trees.

7.3 P2P RDF Repository (RDFPeers)

RDF (Resource Description Framework) is the W3C specification for modeling metadata on the Semantic Web. It is also critical for Grids [26] and Peer-to-Peer systems [21]. RDF makes flexible statements about resources that are uniquely identified by URIs. RDF statements can be distributed on the Web and made by different users. We have developed a scalable P2P RDF repository named RDFPeers to allow each node to store, query and subscribe to RDF statements.

The nodes in RDFPeers form a Chord overlay network. When an RDF triple is inserted into the network, it will be stored at three places by applying a globally-known hash function to its subject, predicate, and object values. Both exact-match and range queries can be efficiently routed to those nodes where the matching triples are known to be stored. The subscriptions for RDF statements are also routed to and stored on those nodes. Therefore, the subscribers will be notified when matching triples are inserted into the network. By building DAT trees on top of RDFPeers, we extend the RDQL language to support aggregate queries, such as *count*, *sum*, *average* and so on. For example, the follow RDQL query returns the number of persons older than 24:

SELECT COUNT(?person), WHERE (?person, <inf:age>, ?age> AND ?age > 24.

In RDFPeers, we define the aggregation latency as the time duration between an aggregate request is issued and the root node yields the aggregated value. Fig. 12 evaluates the aggregation latency of RDFPeers by varying the network size form 32 to 512. This experiment uses UDP-based RPC module on our 8-node cluster with up to 64 RDFPeers nodes per machine. Since the on-demand aggregations in RDFPeers uses a top-down approach, the latency is determined by the tree height of the DAT. Therefore, the latency increases on a log-scale with the network size as shown in Fig. 12.

7.4 Other Important DAT Applications

Besides the above applications, the DAT system also has broad applications in other domains, such as distributed worm signature generation and wide-area network monitoring. We have developed a *WormShield* system [5]that automatically generates worm signatures by aggregating the global fingerprint statistics using DAT trees. In WormShield, a DAT tree is automatically generated for each fingerprint to aggregate its global repetition and address dispersion. In addition, DAT trees can also be applied in wide-area network monitoring systems, such as the MIND [17], to aggregate the global NetFlow statistics from multiple ISP domains.

8 Conclusions

We have presented the DAT algorithms, prototype implementation, performance evaluation and applications on P2P systems and Grids. Our work extends previous methods for distributed information aggregation. Summarized below are four major contributions:

(1) ***Balanced global aggregation schemes :*** We proposed balanced DAT schemes on Chord overlays to aggregate the global information in an efficient and load-balanced fashion.

(2) ***New Chord routing algorithm:*** We developed a balanced Chord routing algorithm to enable the construction of balanced DATs, when nodes are evenly distributed in the identifier space.

(3) ***Prototype DAT and performance results:*** The prototype DAT is available to open research community. Our DAT has been successfully evaluated with good tree properties, message overheads, and load balancing.

(4) ***Wide Applications in P2P and Grid computing:*** We demonstrate that the DAT scheme performs well in Grid resource monitoring, P2P reputation aggregation, P2P RDF repository, distributed worm signature generation, and wide-area network monitoring.

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 Annual ACM Symposium on Theory of Computing* (STOC'03), June 2003.

[2] K. Albrecht, R. Arnold, M. Gahwiler, and R. Wattenhofer, "Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave," in *Proc. of the 4-th International Conference on Peer-to-Peer Computing* (P2P'04), 2004, pp. 227-234.

[3] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, "Estimating Aggregates on a Peer-to-Peer Network," *Computer Science Department, Stanford University, Technical Report*, 2003.

[4] A. G. S. Boyd, B. Prabhakar, and D. Shah, "Randomized Gossip Algorithms," *ACM/IEEE Transactions on Networking*, June 2006.

[5] M. Cai, K. Hwang, J. Pan, and C. Papadopoulos, "Wormshield: Fast Worm Signature Generation with Distributed Fingerprint Aggregation," *IEEE Transaction on Dependable and Secure Computing* (TDSC), submitted December 2005 and revised July, 2006.

[6] M. Cai and M. Frank, "RDFPeers: a Scalable Distributed RDF Repository based on a Structured Peer-to-Peer Network," in *Proc. of the 13th International Conf. on World Wide Web*, 2004.

[7] 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.

[8] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *The International*

Journal of Supercomputer Applications and High Performance Computing, vol. 11, no. 2, 1997.

[9] L. Galanis and D. J. DeWitt., "Scalable Distributed Aggregate Computations through Collaboration," in *Proc. of the 16th Int'l Conf. on Database and Expert Systems Applications*, 2005.

[10] I. Gupta, R. van Renesse, and K. Birman, "Scalable Fault-tolerant Aggregation in Large Process Groups," in *Proc. of the International Conference on Dependable Systems and Networks*, 2001.

[11] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based Aggregation in Large Dynamic Networks," *ACM Transaction on Computer Systems*, vol. 23, no. 3, pp. 219-252, 2005.

[12] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigentrust Algorithm for Reputation Management in P2P Networks," *Proc. of the 12th Int'l Conf.on World Wide Web*, 2003, pp. 640-651.

[13] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based Computation of Aggregate Information," in *Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003, p. 482.

[14] E. Korpela, D. Werthimer, D. Anderson, J. Cobb and M. Lebofsky. "SETI@Home - Massively Distributed Computing for SETI", *Computing in Science & Engineering*, pp. 78-83, January 2001.

[15] 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, pp. 81-92, 2005.

[16] J. Li, and J. Srivastava, "Efficient Aggregation Algorithms for Compressed Data Warehouses," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 14, No. 3, pp. 515-529, May/Jun, 2002.

[17] X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, W. Hong, and G. Iannaccone, "MIND: A Distributed Multi-dimensional Indexing System for Network Diagnosis," in *Proc. of INFOCOM*, 2006.

[18] Y. Liu, L. Xiao, X. Liu, L. M. Ni, X. Zhang, "Location Awareness in Unstructured Peer-to-Peer Systems", *IEEE Trans. on Parallel and Distributed Systems,* Vol. 16, No. 2, Feb 2005, pp. 163 - 174.

[19] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer information System based on the XOR Metric," in *Proc. of the International Workshop on Peer-to-Peer Systems* (IPTPS '02), 2002.

[20] X. Meng, T. Nandagopal, L. Li, and S. Lu, "Contour Maps: Monitoring and Diagnosis in Sensor Networks," *Computer Networks Journal*, 2006.

[21] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch, "Edutella: A P2P Networking Infrastructure based on RDF," in *Proc. of the World Wide Web Conference* (WWW2002), Hawaii, May 2002, pp. 7-11.

[22] L. Ramaswamy, B. Gedik, L. Liu, "A Distributed Approach to Node Clustering in Decentralized Peer-to-Peer Networks," *IEEE Trans. on Parallel and Distributed Systems,* Vol.16, No.9, Sept 2005.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," in *Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (SIGCOMM), 2001.

[24] 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*, Vol. 21, No. 2, pp. 164-206, 2003.

[25] 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), February 2004.

[26] D. D. Roure, N. R. Jennings, N.R. Shadbolt, "The Semantic Grid: Past, Present, and Future", *Proc. of the IEEE*, Vol 93, Issue 3, March 2005, pp. 669-681.

[27] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 2218, 2001.

[28] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (SIGCOMM), 2001.

[29] P. Uppuluri, N. Jabisetti, U. Joshi, Y. Lee, "P2P Grid: Service Oriented Framework for Distributed Resource Management", *Proc. of IEEE Int'l Conference on Services Computing* (SCC'05), 2005.

[30] D. Wagner, "Resilient Aggregation in Sensor Networks," in *Proc. of the 2nd ACM Workshop on Security of Ad hoc and Sensor Networks*, 2004, pp. 78-87.

[31] P. Yalagandula and M. Dahlin, "A Scalable Distributed Information Management System," in *Proc. of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, New York, NY, 2004, pp. 379-390.

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

[33] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: a Fault-tolerant Wide-area Application Infrastructure," in *ACM Computer Communication Review*, Vol. 32, No. 1, 2002, p. 81.

[34] J. Zhao, R. Govindan, and D. Estrin, "Computing Aggregates for Monitoring Wireless Sensor Networks," in *Proc. of the 1st IEEE International Workshop on Sensor Network Protocols and Applications* (SNPA'03), Anchorage, AK, USA, May 2003.

[35] R. Zhou and K. Hwang, "PowerTrust: A Robust and Scalable Reputation System for Trusted P2P Computing," *IEEE Trans.on Parallel and Distributed Systems* (TPDS), accepted March 2006.

[36] Y. Zhu, Y. Hu, "Efficient, Proximity-Aware Load Balancing for DHT-based P2P systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol 16, No. 4, April 2005, pp. 349 - 361.

Appendix A: Proof of Theorem 1

For case (2), it is obvious that $B(i, n) = \log_2(n) - \lceil \log_2(d+1) \rceil = 0$ since (a) $\forall v \in (i, r]$, PARENT(v) $\in (i, r]$, and (b) $\forall v \in (r, i]$, PARENT(v) = FINGER$^+$(v, b) $\in (i, r]$. For case (1), the children of i are its inbound fingers in $(r, i-2^j]$, where $j = \lceil \log_2(d+1) \rceil$. Since $d < 2^{b-1}$, we have $i - 2^j > r$. We split our proof in four disjoint intervals as shown in Fig. 3(a).

(i) $\forall v$ =FINGER$^-$(i, k)$\in (r, i-2^j]$ where $k \geq j+1$, we have PARENT(v) = FINGER$^+$(v, k) = i since FINGER$^+$(v, $k+1$) = $i - 2^{k-1} + 2^k \geq i + 2^j = i + 2^{\lceil \log_2(d+1) \rceil} \geq i + d + 1 = r + 1$.

(ii) $\forall v \in (i-2^j, r-2^j]$, we have PARENT($v$) = FINGER$^+$($v$, j+1) $\in (i, r]$.

(iii) $\forall v \in (r-2^j, i)$, FINGER$^+$(v, j)= $v + 2^{\lceil \log_2(d+1) \rceil - 1} < i - 1 + d + 1 = r$, and FINGER$^+$($v$, j)= $v + 2^{j-1} \geq i - 2^j + 1 + 2^{j-1} = r + 1 - 2^{\lceil \log_2(d+1) \rceil - 1} > r + 1 - (d+1) = i$. Thus, PARENT($v$) = FINGER$^+$($v$, j) $\in (i, r)$.

(iv) $\forall v \in [i, r]$, PARENT(v)$\in (i, r]$.

Figure 3(b) divides the parents of nodes in intervals (i), (ii), and (iii). When $1 < d < 2^{b-1}$, i has j inbound fingers in (ii) and (iii), $\log_2(n) - j$ inbound fingers in (i), and 0 inbound fingers in (iv). Thus, node i has $B(i,$

$n) = \log_2 n - j = \log_2 n - \lceil \log_2(d+1) \rceil$ children in the DAT. When $n < 2^b$, we shrink the identifier space by $d_0 = n/2^b$. This proves $B(i,n) = \log_2(n) - \lceil \log_2 (d/d_0 + 1) \rceil$.

Appendix B: Solution of Equation (1)

We solve the equation in two cases: (a) $d=2^k-2$, and (b) $d=2^k-2+\delta$, where $k=1,2,...,b-1$ and $1 \le \delta < 2^k$. For case (a), we have $x=2^k-2+2^k$. Therefore, $g(x)=k=\log_2((x+2)/2)$. For case (b), since $x=2^k-2+\delta+2^{k+1}$ $=3 \cdot 2^k-2+\delta$, then $g(x) = \lceil \log_2(d+2) \rceil = \lceil \log_2(2^k+\delta) \rceil = \lceil \log_2((x+2)/3+2\delta/3) \rceil$. Also, since $1 \le \delta \le 2^k$, we have

$$\left\lceil \log_2(\frac{x+2}{3}) \right\rceil \le \left\lceil \log_2(\frac{x+2}{3} + \frac{2\delta}{3}) \right\rceil = \left\lceil \log_2(\frac{3 \times 2^k + \delta}{3} + \frac{2\delta}{3}) \right\rceil = \left\lceil \log_2(2^k + \delta) \right\rceil$$

$$< \left\lceil \log_2(2^k + 2^k) \right\rceil = k + 1 < \left\lceil \log_2(2^k + \frac{\delta}{3}) \right\rceil + 1 = \left\lceil \log_2(\frac{x+2}{3}) \right\rceil + 1$$

Thus, the solution for case (b) is $g(x)=\lceil \log_2((x+2)/3) \rceil$. To normalize $g(x)$ for these two cases, we prove that $\log_2((x+2)/2)=\lceil \log_2((x+2)/3) \rceil$ when $x=2^{k+1}-2$ as follows:

$$\left\lceil \log_2(\frac{x+2}{3}) \right\rceil = \left\lceil \log_2 2^{k+1} - \log_2 3 \right\rceil = \left\lceil k + 1 - \log_2 3 \right\rceil = k = \log_2 \frac{x+2}{2}.$$

Therefore, we can derive $g(x)=\lceil \log_2((x+2)/3) \rceil$ from Eq.(1) for both cases.

Biographical Sketches:

Min Cai received his BS and MS degrees in Computer Science from Southeast University, China, in 1998 and 2001, respectively. He joined IBM Research, Beijing, in 2001, where he worked on multimedia networking. He is currently a Ph. D. candidate in Computer Science Department at the University of Southern California. His research interests include intrusion detection, peer-to-peer and grid computing, semantic web and web services technologies. His email address is *mincai@usc.edu*.

Kai Hwang is a Professor of Electrical Engineering and Computer Science at USC. He received the Ph.D. degree from the University of California, Berkeley. An IEEE Fellow, he specializes in computer architecture, parallel processing, Internet and wireless security, P2P, Grid and distributed computing systems. He has published over 200 original scientific papers and 7 popular books in theses area. Presently, he leads the USC GridSec project in developing security-binding techniques and distributed defense systems against worms and DDoS attacks for trusted Grid, P2P, and Internet computing. Contact him at *kaihwang@usc.edu* or visit the web site: *http://gridsec.usc.edu/Hwang.html*.