Research Challenges for a Scalable Distributed Information Management System

Praveen Yalagandula and Mike Dahlin
Laboratory for Advanced Systems Research
Department of Computer Sciences
The University of Texas at Austin

Abstract

A Scalable Distributed Information Management System (SDIMS) that *aggregates* information about large-scale networked systems can serve as a basic building for a broad-range of large-scale distributed applications simplifying the design, development, and deployment of such services. In this document, we outline four key requirements such an aggregation system should satisfy to be useful as a general middleware building block – scalability with both nodes and data attributes, flexibility to accommodate broad range of services, administrative autonomy and isolation for availability and security, and robustness to reconfigurations in the system. We propose a new aggregation framework that leverages Distributed Hash Tables (DHTs) and a new aggregation abstraction that builds on a previously proposed abstraction in Astrolabe. We also present details of several significant applications that we propose to build on top of SDIMS.

1 Introduction

The goal of this research is to design and build a Scalable Distributed Information Management System (SDIMS) that *aggregates* information about large-scale networked systems and that can serve as a basic building block for a broad range of large-scale distributed applications. Monitoring, querying, and reacting to changes in the state of a distributed system are core components of applications such as system management [11, 36, 71, 80, 84, 93], service placement [35, 94], grid scheduling [4, 7, 8, 17, 21, 47, 34, 32, 54, 76], data sharing and caching [61, 70, 74, 79, 83, 100], sensor monitoring and control [48, 56], multicast tree formation [14, 15, 86, 78, 81], and naming and request routing [16, 19]. We therefore speculate that an SDIMS in a networked system would provide a "distributed operating systems backbone" and facilitate the development and deployment of new distributed services.

For a large scale information system, *hierarchical aggregation* is a fundamental abstraction for scalability. Rather than expose all information to all nodes, hierarchical aggregation allows a node to access detailed views of nearby information and summary views of global information. In an SDIMS based on hierarchical aggregation, different nodes can therefore receive different answers to the query "find a [nearby] node with at least 1 GB of free memory" or "find a [nearby] copy of file foo." A hierarchical system that aggregates information through reduction trees [56, 86] allows nodes to access information they care about while maintaining system scalability.

1.1 Requirements

To be used as a general middleware building block, an SDIMS should have four properties – (1) *Scalability* with respect to both nodes and attributes, (2) *Flexibility* to accommodate a broad range of applications, (3) *Administrative autonomy and isolation* for availability and security, and (4) *Robustness* to reconfigurations in the system.

Scalability The system should accommodate large numbers of participating nodes, and it should allow applications to install and monitor large numbers of data attributes. Enterprise and global scale systems might have tens of thousands to millions of nodes and these numbers will increase as desktop machines give way to larger numbers of smaller devices. Similarly, we hope to support many applications and each application may track several attributes (e.g., the load and free memory of a system's machines) or millions of attributes (e.g., which files are stored on which machines).

Flexibility The system should have flexibility to accommodate a broad range of applications and attributes. For example, *read-dominated* attributes like *numCPUs* rarely change in value, while *write-dominated* attributes like *cpuLoad* change quite often. An approach tuned for read-dominated attributes will suffer from high bandwidth consumption

when applied to write-dominated attributes. Conversely, an approach tuned for write-dominated attributes may suffer from unnecessary query latency or imprecision for read-dominated attributes. Therefore, an SDIMS should provide a flexible mechanism that can efficiently handle different types of attributes and either leave the policy decision of tuning read and write propagation to the application installing an attribute or autonomously self-tune it strategies to suit the observed workload characteristics.

Autonomy and Isolation In a large computing platform, it is natural to arrange nodes in an organizational, administrative, or virtual-organization [33] hierarchy. An SDIMS should support administrative autonomy so that, for example, a system administrator can control what information flows out of her machines and what queries may be installed on them. And, an SDIMS should provide isolation in which queries about a domain's information can be satisfied within the domain so that the system can operate during disconnections and so that an external observer cannot monitor or affect intra-domain queries.

Robustness The system must be robust to node failures and disconnections. An SDIMS should adapt to reconfigurations in a timely fashion and should also provide mechanisms so that applications can exploit the tradeoff between the cost of adaptation versus the consistency level in the aggregated results and the response latency when reconfigurations occur.

1.2 Our approach

We propose to develop a working SDIMS prototype that meets the above requirements and to refine this prototype framework by developing several significant applications over it.

Our SDIMS prototype will explore a number of key research issues for meeting the listed goals. It will build on the emerging literature of distributed hash tables (DHTs) [5, 40, 43, 49, 53, 58, 59, 61, 68, 70, 74, 79, 100] to provide a scalable, self-managing substrate for aggregation by mapping different attributes to the myriad different virtual trees commonly present in DHTs' internal structures. Although this basic strategy appears promising, our initial analysis suggests that significant research challenges must be overcome to adapt DHT technology to address the problem of scalable aggregation. We will adapt the aggregation API to support *scalability* by ensuring that aggregation requests expose sufficient parallelism to the underlying architecture to allow us to efficiently and scalably map them to different paths within the underlying DHT, and we will also study ways to efficiently answer composite queries that span multiple aggregation trees. We will develop an aggregation API that provides *flexibility* by allowing applications to control the propagation of information through the system to adapt to the read/write frequency of different applications and to adapt to spatial and temporal heterogeneity of applications' access patterns, and we will also explore mechanisms and policies for making this adaptation self-tuning so that it requires less intervention by the application programmer. We will adapt the aggregation API and DHT routing protocols to ensure *administrative autonomy* so that queries across a set of administratively related nodes can be satisfied by just those nodes despite the changes in the DHT substrate. Finally, we will address the problem of *robustness.* Although the literature of DHTs provides ways for a DHT to reconfigure itself in the face of failures, we must go further and ensure that the aggregation abstraction defined over trees internal to a DHT continue to return sensible results in the face of reconfigurations of the underlying DHT. We will explore how to combine replication in space and in time in order to allow an application whose functionality is intimately tied to the underlying DHT architecture to continue to function despite DHT reconfigurations.

The second major aspect of our work will be to construct three significant real-world applications over this SDIMS middleware. First, we will use SDIMS as a control backplane for a new enterprise-scale file system that provides Partial Replication, Arbitrary Consistency, and Topology Independence (PRACTI) [27]. In this environment, SDIMS will provide (1) a data location service for routing requests to the nearest current replica of a given object, (2) a monitoring backplane to ensure that a minimum number of replicas is maintained at all times, (3) a substrate for rendezvousing to form multicast trees for the self-tuning [51, 89] propagation of updates to nodes interested in specific sets of files, and (4) a monitoring system to track object read and write frequencies to support massive speculative replication [88]. A key challenge in this environment will be dealing with a heterogeneous collection of fixed-location servers and intermittently-connected mobile devices. Second, we will construct a grid information system that exports the MDS-2 [20] or GIS [2] interface but that internally makes use of SDIMS for improved scalability, flexibility, simplicity of administration, and robustness. Using an existing grid information system API will allow a broad range of existing applications and services to make immediate use of our improved abstraction as well as enable new, more demanding grid applications. We will focus particular attention on developing a grid scheduler [4, 7, 8, 17, 21, 47, 34, 32, 54, 76] based on the Community Scheduling Framework (CSF) [77] suitable for deployment by the Texas Advanced Computing Center across both campus-scale and state-scale grids. This scheduler will use SDIMS both to

monitor system health and performance within clusters and to aggregate information across federations of clusters to guide global scheduling decisions. Third, we will use SDIMS to develop novel network monitoring applications that, for example, detect unusual global patterns such as a distributed denial of service attack (DDOS) by aggregating key traffic pattern events from sensors implemented on programmable routers [42] across the network.

Our SDIMS development and application prototyping efforts will be closely tied and we will iteratively refine the SDIMS abstraction and implementation to address limitations or opportunities exposed by our efforts to construct applications over it.

This research will result in a new abstraction that acts as a "distributed systems backplane", that is qualitatively more scalable, flexible, simple to administer, and robust than existing solutions, and that dramatically simplifies the development of new classes of large-scale information systems.

2 Background

In this section, we first formally define our aggregation abstraction and then discuss key related work on which we build.

2.1 Aggregation abstraction

Aggregation is a natural abstraction for a large-scale distributed information system because aggregation can support scalability by allowing a node to view detailed information about the state near it and progressively coarser-grained summaries about progressively larger subsets of a system's data [86].

Our aggregation abstraction is defined across a tree spanning all nodes in the system. Each physical node in the system is a leaf and each subtree represents a logical group of nodes. Note that logical groups can correspond to virtual organizations [33] (e.g., a multi-agency task force responding to a natural disaster), traditional administrative unit (e.g., a department in a company or a university in a system), a DNS domain or subdomain, or groups of nodes within a domain (e.g., a /28 subnet with 14 hosts on a LAN in the CS department).

Each physical node has *local data* stored as a set of $(attributeType, attributeName, value)$ tuples such as *(configuration, numCPUs, 16), (mcast membership, session foo, yes)*, or *(file stored, foo, myIPaddress)*. The system associates an *aggregation function* f_{type} with each attribute type, and for each level-i subtree T_i in the system, the system defines an *aggregate value* $V_{i,type,name}$ for each *(attributeType, attributeName)* pair as follows. For a (physical) leaf node T_0 at level 0, $V_{0,type,name}$ is the locally stored value for the attribute type and name or NULL if no matching tuple exists. Then the aggregate value for a level-i subtree T_i is the aggregation function for the type computed across the aggregate values of each of T_i's k children: $V_{i,type,name} = f_{type}(V^0_{i-1,type,name}, V^1_{i-1,type,name}, \ldots, V^{k-1}_{i-1,type,name})$.

Although SDIMS allows arbitrary aggregation functions, it is often desirable that aggregation functions satisfy the *hierarchical computation* property [56]: $f(v_1, ..., v_n) = f(f(v_1, ..., v_{s_1}), f(v_{s_1+1}, ..., v_{s_2}), ..., f(v_{s_k+1}, ..., v_n))$, where v_i is the value of an attribute at node i. For example, the average operation, defined as $avg(v_1, ..., v_n) = 1/n. \sum_{i=0}^{n} v_i$, does not satisfy the property. But, if instead an attribute type stores values as tuples $(sum, count)$ and defines the aggregation function as $avg(v_1, ..., v_n) = (\sum_{i=0}^{n} v_i.sum, \sum_{i=0}^{n} v_i.count)$, the attribute satisfies the hierarchical computation property. Note that an application wrapper then must compute the average from the aggregate sum and count values.

Notice that this definition of the aggregation abstraction leaves a number of degrees of freedom to an implementation to decide who computes a given subtree's aggregates (e.g., one node or many nodes) and when they are computed (e.g., eagerly on updates or lazily on reads). As described below, in our proposed SDIMS system an internal non-leaf node N_i of the aggregation tree is simulated by one or more physical nodes that belong to the subtree for which the N_i is the root, and the system will use these degrees of freedom to adjust replication and propagation to improve robustness and efficiency.

Finally, note that for a large-scale system, it is difficult or impossible to insist that the aggregation value returned by a probe corresponds to the function computed over the current values at the leaves at the instant of the probe. Therefore our proposed implementation provides only weak consistency guarantees – specifically eventual consistency as defined in [86]. Section 3.4 discusses issues relating to consistency in more detail.

2.2 Previous work

We draw inspiration most directly from two main bodies of previous work: distributed monitoring systems and Distributed Hash Tables (DHTs).

Distributed monitoring. Astrolabe [86] is a robust information management system from which we draw most heavily. Astrolabe provides the abstraction of a single logical aggregation tree that mirrors a system's administrative hierarchy for autonomy and isolation. It provides a general interface for installing new aggregation functions and provides eventual consistency on its data. Astrolabe is highly robust due to its use of an unstructured gossip protocol for disseminating information and its strategy of replicating all aggregated attribute values for a subtree to all nodes in the subtree. This combination allows any communication pattern to yield eventual consistency and allows any node to answer any query using local information. This high degree of replication, however, may limit the system's ability to accommodate large numbers of attributes. Also, although the approach works well for read-dominated attributes, an update at one node can eventually affect the state at all nodes, which may limit the system's flexibility to support write-dominated attributes.

Other related information management projects include Willow [87], Cone [9], DASIS [1], and SOMO [99]. Willow, DASIS and SOMO build a single tree for aggregation. Cone builds a tree per attribute and requires a total order on the attribute values.

Whereas above mentioned projects propose a general information collection and management system, several academic [36, 56, 93] and commercial [84] distributed monitoring systems have been designed to particularly monitor the status of large networked systems. Some of them are centralized where all the monitoring data is collected and analyzed at a single central host. Ganglia [36, 60] uses a hierarchical system where the attributes are replicated within clusters using multicast and then cluster aggregates are further aggregated along a single tree. Sophia [93] is a distributed monitoring system, currently deployed on Planet-Lab [67], and is designed around a declarative logic programming model where the location of query execution is both explicit in the language and can be calculated in the course of evaluation. This research is complementary to our work; the programming model can be exploited in our system too. TAG [56] collects information from a large number of sensors along a single tree.

In the grid community, a number of existing information systems such as GIS [2], MDS-2 [20], R-GMA [12], and Hawkeye [44] each provide a core distributed information management system designed to support a range of applications and services such as scheduling, replica selection, service discovery, and system monitoring. All of these systems use a client-server model in which *Information Providers* collect or generate data and supply this data to *Information Servers* through which applications access data.[1] In some of these systems, scalability beyond this basic client-server model is provided by time to live (TTL) based caching, pushing of updates, and arrangement of Information Servers into cache hierarchies.

DHTs. Recent research in peer-to-peer structured networks resulted in Distributed Hash Tables (DHTs) [5, 40, 43, 49, 53, 58, 59, 61, 68, 70, 74, 79, 100]—a data structure that scales with the number of nodes and that distributes the read-write load for different queries among the participating nodes.

It is interesting to note that the observation that DHTs internally provide a scalable forest of reduction trees is not new. Plaxton et al.'s [68] original paper describes not a DHT abstraction, but a system for hierarchically aggregating and querying object location data in order to route requests to nearby copies of objects. Many systems—building upon both Plaxton's bit-correcting strategy [74, 100] and upon other strategies [61, 70, 79]—have chosen to hide this power and export a simple and general distributed hash table abstraction as a useful building block for a broad range of distributed applications. Some of these systems internally make use of the reduction forest not only for routing but also for caching [74], but for simplicity, these systems do not generally export this powerful functionality in their external interface. Our goal is to develop and expose the internal reduction forest of DHTs as a similarly general and useful abstraction and building block. Dabek et al. [25] propose common APIs (KBR) for structured peer-to-peer overlays that facilitate the application development that is independent from the underlying overlay. While KBR facilitates the deployment of our abstraction on any DHT implementation that supports the KBR API, it does not provide any interface to access the list of children for different prefixes. It therefore seems appealing to develop an SDIMS abstraction that exposes this internal functionality in a general way so that scalable trees for aggregation can be considered a basic system building block alongside the distributed hash tables.

Although object-location application are a predominant target for DHTs, several other applications like multicast [14, 15, 78, 81], file storage [23, 52, 73], and DNS [19] are also built using DHTs. All of these applications implicitly perform aggregation on some attribute, and each one of them must be designed to handle any reconfigurations in the underlying DHT. With the aggregation abstraction provided by our system, designing and building of such applications would become easier.

[1] Rather than use the terminology for different components used by different systems, we follow Zhang et al.'s [98] terminology for the common features of these systems.

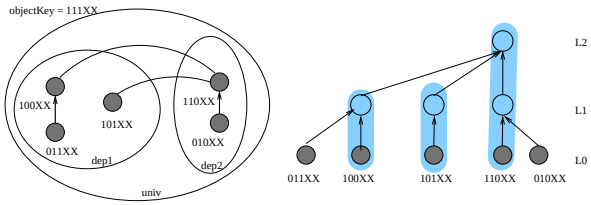

Figure 1: Example DHT tree across a collection of five nodes for objectKey 111XX. In the left figure, filled circles represent nodes and larger circles indicate node membership in DNS domains. In the right figure, filled circles represent physical nodes, empty circles represent virtual nodes, and shaded regions show which virtual nodes are simulated by which physical nodes.

3 SDIMS Research and Development

The SDIMS architecture maps a forest of aggregation trees to a collection of nodes using distributed hash table (DHT) techniques. As Figure 1 illustrates, distributed hash tables assign each node in the system a random *nodeKey* and use a cooperative peer-to-peer algorithm to route a request with some *objectKey* to the nodeKey that matches the objectKey in the most bits.[2] For example, Plaxton's algorithm has each node N_1 forward a request to a node N_2 whose nodeKey matches the request's objectKey in more bits than N_1's nodeKey. Note that if all of the nodes were to issue requests with the same objectKey, the union of the routes taken by the requests would form a tree with all of the nodes acting as leaves, some of the nodes acting as interior nodes, and one node acting as root. Also note that the set of routes taken by different objectKeys represents a forest of trees, with each node acting as the root of an approximately n-node subtree for approximately $\frac{1}{n}$ of the objectKeys.

The simple idea at the core of SDIMS is to exploit a DHT's forest of trees as a collection of reduction trees for data aggregation: trees are a natural framework for aggregation and DHT constructions should result in good load balancing and locality across these trees.

At first glance, it might appear obvious that simply combining DHTs with an aggregation abstraction will result in an SDIMS. However, for SDIMS to form the basis of a general distributed systems control backplane, it must address four questions: (i) How to provide flexibility in the aggregation to accommodate different application requirements?, (ii) How to scalably map different attributes to different aggregation trees within a DHT mesh? (iii) How to adapt a global, flat DHT mesh to satisfy the required autonomy and isolation properties? and (iv) How to provide good robustness without unstructured gossip and total replication? Our research will address these issues.

3.1 Flexible Computation and Propagation

The definition of the aggregation abstraction permits a continuous spectrum of computation and propagation strategies ranging from lazy aggregate computation and propagation on reads to an aggressive immediate computation and propagation of aggregate values on writes. In Figure 2, we illustrate both these extreme strategies and an intermediate strategy. Under the lazy *Update-Local* computation and propagation strategy, an update (aka write) only affects local state. Then, a probe (aka read) that reads a level-i aggregate value is sent up the tree to the issuing node's level-i ancestor and then down the tree to the leaves. The system then computes the desired aggregate value at each layer up the tree until the level-i ancestor that holds the desired value. Finally, the level-i ancestor sends the result down the tree to the issuing node. In the other extreme case of the aggressive *Update-All* immediate computation and propagation on reads [86], when an update occurs, changes are aggregated up the tree, and each new aggregate value is broadcast to all of a node's descendants. In this case, each level-i node not only maintains the aggregate values for the level-i subtree but also receives and locally stores copies of all of its ancestors' level-j ($j > i$) aggregation values. Also, a leaf satisfies a probe for a level-i aggregate using purely local data. In an intermediate *Update-Up* strategy, the root of each subtree maintains the subtree's current aggregate value, and when an update occurs, the leaf node updates its local state and passes the update to its parent, and then each successive enclosing subtree updates its aggregate value and passes the new value to its parent. This strategy satisfies a leaf's probe for a level-i aggregate value by sending the probe up to the level-i ancestor of the leaf and then sending the aggregate value down to the leaf. Finally, notice that other strategies also exist. In general, an Update-Up`k`-Down`j` strategy aggregates up to the `k`th level and propagates the aggregate values of a node at level l (s.t. $l \leq$ `k`) downwards for `j` levels.

[2] Some DHT algorithm route objectKeys to the node with the highest nodeKey value that does not exceed objectKey.

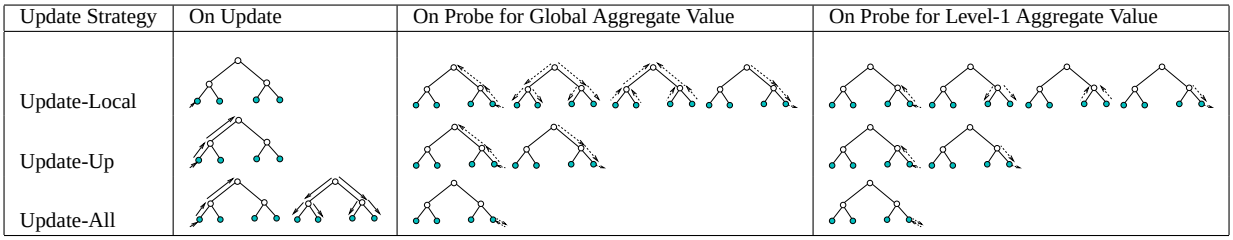

Figure 2: Illustration of policies supported by SDIMS's flexible API

Although this large design space exist for the aggregation computation, previous systems like Astrolabe [86], Ganglia [36] and DHT based systems [70, 74, 79, 100] chose to implement a single static propagation mechanism. For example, Astrolabe implements an Update-All strategy for robustness while most DHT based systems chose Update-Up.

Why should an SDIMS provide flexibility? For SDIMS to to be a general abstraction that is useful to a wide range of applications, it must provide a wide range of flexible computation and propagation strategies. In particular, it should accommodate *application heterogeneity* where different applications or attributes have different access patterns; for example a *read-dominated* attribute like *numCPUs* rarely changes in value, while a *write-dominated* attribute like *numProcesses* changes quite often. An aggregation strategy like Update-All works well for read-dominated attributes but suffers high bandwidth consumption when applied for write-dominated attributes. Conversely, an approach like Update-Local works well for write-dominated attributes but suffers from unnecessary query latency or imprecision for read-dominated attributes. Furthermore, SDIMS should accommodate *spatial heterogeneity* where an attribute is accessed with different access patterns in different regions of the system or by different nodes scattered throughout the system. For example, a particular data file might be of interest to applications running across a set of geographically close nodes (e.g., a cluster), so a file-directory system should optimize the performance of queries about that file from that region of the system. And SDIMS should accommodate *temporal heterogeneity* when an attribute is accessed differently at different times during the execution of an application or across hours or days as application mixes change.

Research Agenda. We propose to develop mechanisms and policies to control flexible aggregation.

On the mechanism side, to provide the level of control required we must decompose the aggregation abstraction into a distributed interface that provides appropriate control of installation, updates, and probes and that allows different policies for different attributes and for different nodes. We have developed an initial design in which (1) *installation requests* that install a new attribute type specify the aggregation function for the type and also control the default upward propagation of updates and downward propagation of computed aggregates, (2) *update requests* that update the value for an (*attributeType, attributeName*) pair can override the default up/down propagation, and where (3) probe requests that retrieve an aggregate value for a particular (*attributeType, attributeName, levelOfTree*) can specify whether to return just the current value or whether to propagate new values down towards the requester when an update triggers a change to the aggregate value (e.g., to support continuous queries). The mechanisms further allow an attribute's "up" and "down" values to be reset on a node-by-node basis so that the system can cope with heterogeneous demands across time or across space.

The simulation results in Figure 3 illustrates the potential benefits of this flexibility. We simulate a system with 4096 nodes arranged in a domain hierarchy with branching factor (bf) of 16 and install several attributes with different *up* and *down* parameters. We plot the average number of messages per operation incurred by different attributes for a wide range of read-to-write ratios of the operations. This graph clearly demonstrates the need for a wide range of computation and propagation strategies for a middleware that seeks to support heterogeneous services: different settings yield order of magnitude differences in per-request overheads. Furthermore, not only does this flexibility to control propagation allow applications to optimize overhead, it also allows applications to reduce read latency at the cost of additional overhead when responsiveness is more important than efficiency.

Given these mechanisms, some applications will wish to explicitly control the propagation of updates and probes, but other applications will benefit from *self-tuning adaptation* policies. Such policies will be most useful when access patterns to different attributes within an application vary over time (temporal heterogeneity) or vary across nodes (spatial heterogeneity). Instead of requiring applications to keep track of the exact read-to-write ratios, which we expect to

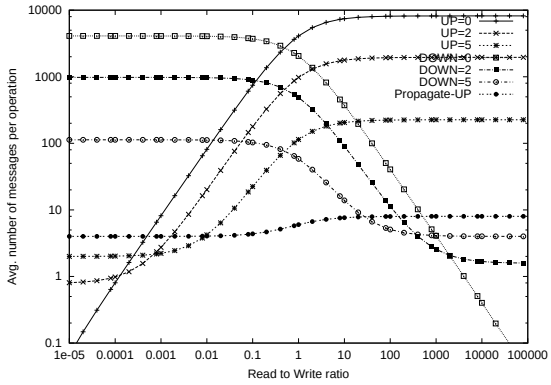

Figure 3: Flexibility of our approach. With different UP and DOWN values in a network of 4096 nodes for different read-write ratios.

be dynamic, an SDIMS system should adapt to changing read-to-write ratios by dynamically adapting the computation and propagation modes to conserve computation and communication resources while still meeting application targets for responsiveness or availability.

A final challenge is providing consistency guarantees beyond best-effort eventual consistency. In particular, an application should be able to trade precision for performance. SDIMS queries should be capable of providing a range of consistency guarantees at a range of costs, such as best-effort eventual consistency to specific bounds on the numerical error or staleness of a query result. We believe that past efforts in single tree aggregation [64, 63] or peer-to-peer data replication [97] can be adapted to the requirements of SDIMS, but a challenge is reframing these abstractions to support the more flexible update/probe API we envision.

3.2 Scalability

One of the key requirements of an SDIMS system is that it should be able to scale with both the number of nodes and the number of attributes. Enterprise and global scale systems might have tens of thousands to millions of nodes and these numbers will increase as desktop machines give way to larger numbers of smaller devices. Similarly, we hope to support many applications and each application may track several attributes (e.g., the load and free memory of a system's machines) or millions of attributes (e.g., which files are stored on which machines).

Compared to past aggregation systems, we believe SDIMS can achieve scalability with respect to both nodes and attributes by (1) leveraging DHT technologies to create load balanced aggregation trees, (2) optimizing the system to minimize the cost of handling *sparse attributes*, and (3) developing query planning strategies for dealing with composite queries.

First, while some previous distributed information management systems like Astrolabe [86] and Ganglia [36] aggregate all attributes on a single aggregation hierarchy and others such as MDS-2 [20] and GIS [2] rely on "flat" all-to-all communication among clusters, we leverage Distributed Hash Tables to construct multiple aggregation trees and aggregate different attributes on different trees to achieve scalability with both nodes and attributes. A single tree is unscalable with attributes as the number of aggregations that the root has to perform grows linearly with the number of attributes and the number of updates or probes to those attributes. By aggregating different attributes along different trees, the load of aggregation is split across multiple nodes. In particular, we propose to hash an attribute's name and type and use the resulting key's DHT tree as the aggregation tree for that key. Further refinements should address limiting each node's in-degree to avoid overloading nodes by giving them unexpectedly large numbers of children in DHT formation. For example, RANCH [53], SkipNet [43], and Skip Graph [5] attain an $O(\log n)$ indegree through linear ordering of the nodes and enforcing rules on the prefix pointers that a node can choose; it may be possible to adapt these approaches for SDIMS tree formation.

Second, an SDIMS must optimize its handling of *sparse attributes* to achieve scalability to large numbers of attributes. We expect most systems to have a few dense attributes that are accessed by most or all of the nodes but many sparse attributes that are of more narrow interest. We propose to restrict the scope of sparse attribute propagation by using the flexible propagation mechanisms described above to control downward propagation of updates to only the nodes interested in the updates. Furthermore, in contrast with previous hierarchical systems [36, 86], we have defined

a new aggregation abstraction that specifies both an attribute type and attribute name and associates an aggregation function with a type rather than just specifying an attribute name and associating a function with a name. Installing a single function that can operate on many different named attributes matching a specific type enables our system to efficiently handle applications that install a large number of attributes with same aggregation function. For example, to construct a file location service, our interface allows us to install a single function that computes an aggregate value for any named file (e.g., the aggregate value for the (function, name) pair for a subtree would be the ID of one node in the subtree that stores the named file). Conversely, Astrolabe copes with such attributes by congregating them into a single set and having aggregation functions compute on such sets; Astrolabe also suggests that scalability can be improved by representing such sets with Bloom filters [10]. Exposing names within a type provides at least two advantages. First, when the value associated with a name is updated, only the state associated with that name need be updated and (potentially) propagated to other nodes. Second, splitting values associated with different names into different aggregation values allows our system to leverage Distributed Hash Tables(DHT) to map different names to different trees.

Third, we will explore ways to efficiently handle composite queries that span multiple attributes such as a probe like *find a nearest machine with load less than 20 percent and has more than 2 GB of memory*. The load balancing and scalability provided by hashing different attributes to different trees makes composite queries more complex to handle than simpler schemes that propagate all information via a single tree; composite queries must now gather data from multiple trees and combine the results. If query compositions are known in advance, then attributes can be grouped and can be aggregated along one tree. For example, load and memory of machines can be aggregated along one tree if queries as in the above example are very common. But by grouping extensively, we lose the property of load balancing. This tradeoff presents a fundamental limitation of distributing attributes across trees. One possible approach is to leverage ongoing efforts by other researchers to provide the relational database abstraction on DHTs [46, 38] and to adapt query planning techniques to this environment. As a simple example, queries with OR and AND operations might be handled as follows: (1) a OR b: Walk along trees corresponding to both attributes a and b; (2) a AND b: Guess the smaller of the trees corresponding to a and b, and compute the predicate along the tree. In the latter case, two approaches can be used to determine the size of the trees: (a) Along with the computation of the aggregation function for an attribute, maintain a count of the number of contributing nodes or (b) Use statistical sampling techniques – randomly choose a small percentage of nodes and evaluate the attributes. Given these building blocks, for handling general logical expressions, one can convert the logical expressions to their Disjunctive Normal Forms (DNF) and use above AND operation for each conjunctive term.

We believe that the combination of DHTs and careful handling of sparse attributes is crucial and we believe that by combining these approaches, a node's work should increase linearly with the average number of attributes a node reads or writes and with the log of the number of nodes in the system, resulting in good scalability both with attributes and nodes. Quantifying the scalability of such systems under more complex queries, however, will require gaining experience with real-world workloads.

3.3 Autonomy

The property of *autonomy* allows applications to restrict some SDIMS updates and probes to collections of machines corresponding to administrative units such as DNS subdomains, IP subnets, or organizational units (e.g., a floor of a building, a team in a department, a department in a company, or a virtual organization spanning groups from multiple companies). It is important that SDIMS support autonomy for three reasons: First, autonomy is important for availability—a relatively common failure pattern is for a collection of machines to become disconnected from the internet but remain connected to one another [3, 26]; it is therefore highly desirable that a department be able to continue to access SDIMS for queries about state that resides within the department even when the department's network connection to the rest of the world is down. Second, autonomy is important for security—an organization must be able to prevent information about certain installed attributes, queries, or probes from "leaking" out of the organization, so it must be able to restrict some SDIMS requests to within organizational boundaries. Third, constraining aggregation to follow administrative structure is important for delivering meaningful results to some types of queries. For example, in a Grid scheduling application, one could imagine a policy that preferentially seeks to assign jobs to "fixed cost" in-organization computing resources before farming jobs out to pay-per-use external computing resources [35].

We propose to develop techniques to support autonomy in SDIMS by (1) developing algorithms for *Autonomous Distributed Hash Tables* (ADHTs) and (2) extending the API to and implementation of our middleware to expose administrative information to the aggregation abstraction.

To conform to the administrative autonomy requirement, an ADHT should satisfy two properties:

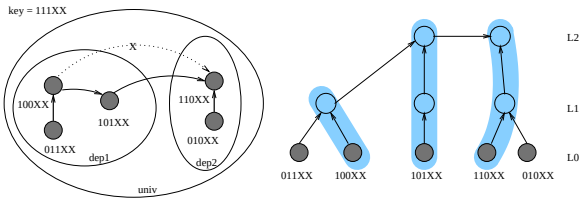

Figure 4: In contrast with the DHT system illustrated in Figure 1, our Autonomous DHT algorithm satisfies the isolation property.

1. Path Locality : Search paths should always be contained in the smallest possible domain.
2. Path Convergence : Search paths for a key from two different nodes in a domain should converge at a node in the same domain.

Although existing DHTs can be adapted to provide some support for path locality by including information about administrative structure in the proximity metric they use to select parent/child pairings [13, 39], these systems do not guarantee path convergence. For example, Figure 1 illustrates a simple case where Pastry's bit-correcting routing scheme cannot ensure convergence even if the proximity metric is tweaked to consider administrative structure: because two nodes within one domain match the specified key in the same maximum number of bits, the parent that aggregates all values in the domain must lie outside of the domain. SkipNet [43] solves a related problem by providing domain restricted routing where a key search can be limited to a specified domain. Unfortunately, SkipNet is not appropriate for SDIMS because it sacrifices the aggregation tree abstraction property of DHTs as its domain constrained routing might touch a node more than once (as it searches forward and then backward to stay within a domain).

To provide the new ADHT abstraction required by SDIMS, the system's route table construction algorithm must provide a single exit point in each domain for a key and its routing protocol should route keys along intra-domain paths before routing them along inter-domain paths. Here we outline modifications to Pastry's route table construction and key-routing protocols achieve these goals; we hypothesize that similar transformation could be applied to other existing DHT implementations to transform them into ADHT systems. Figure 4 illustrates how our ADHT algorithm routes towards the node with nodeId $101XX$ for key $111XX$. In the ADHT, each node maintains a separate leaf set for each domain it is part of; in contrast, Pastry's DHT construction maintains a single leaf set for all the domains. Note that this change increases the number of neighbors that each node tracks to $(2^b) * \lg_b n + c.l$ from $(2^b) * \lg_b n + c$ in unmodified Pastry, where b is the number of bits in a digit, n is the number of nodes, c is the leafset size, and l is the number of domain levels. The algorithm for populating each node's routing table is similar to Pastry with the following difference: it uses hierarchical domain proximity as the primary proximity metric (two nodes that match in i levels of a hierarchical domain are more proximate than two nodes that match in fewer than i levels of a domain) and network distance as the secondary proximity metric (if two pairs of nodes match in the same number of domain levels, then the pair whose separation by network distance is smaller is considered more proximate).

Our research agenda for fulfilling this ADHT vision includes completing development of an ADHT algorithm for Pastry, generalizing this approach to other DHTs such as the standard DHT API recently proposed [22], quantifying the increase in overhead and reduction in network efficiency caused by ADHT's additional routing constraints, and extending the low-level implementation of the ADHT algorithm to integrate securely with existing secure node ID systems such as secure DNS and distributed directories to prevent unauthorized nodes from joining a domain. Once an ADHT has been developed, we believe that the SDIMS abstraction and interfaces can be extended to support autonomy on installs, updates, and probes. Note that in addition to restricting the propagation of information out of a domain due to a domain's own install/update/probe commands, the system must also prevent attributes installed by other domains from accessing and aggregating aspects of local state that are not intended for dissemination; adding an access control list to leaf nodes' attributes is one way to enforce such restrictions [85].

3.4 Robustness

SDIMS faces two key challenges to robustness. First, to scale to a large number of nodes distributed across a network, an SDIMS system must cope with an increasing rate of node failures as well as an increasing rate of site disconnections when a collection of nodes fails or becomes disconnected from the rest of the system. Although DHT systems have explored strategies for improving robustness by replicating the root values for a key to several nearby (in key space)

nodes [18, 23, 55, 73], SDIMS faces a more difficult problem because its abstraction exposes not only the value at a key's root but values computed along the path from each node to the root. Second, SDIMS aims to scale to large numbers of attributes, which increases the amount of per-node state that must be reconstructed on a failure.

Our approach for robustness across failures will follow three tracks: masking and isolating failures at the DHT level, replication (of aggregation information) in space, and replication in time.

We will measure and improve the stability of DHT algorithms to reduce the change in the internal structure of the DHT tree over time. Most past DHT applications have stored application-visible per-key state at the root of the DHT trees, so they are free to adjust internal topology when, for example, measured round trip times between nodes change. We will improve the stability of DHTs internal structure by developing algorithms to weigh the cost/benefit trade-offs of improving routing efficiency versus redistributing internal application-visible state. Also, to exploit the incremental state reconstruction algorithms outlined below, we will develop DHT algorithms that retain their structure over long periods of time so that if a node fails and then rejoins the system, it will tend to rejoin in the same position it previously held and will therefore be able to re-use much of its prior application-visible state. Finally, we will exploit ADHT's properties to isolate topology changes so that reconfigurations in one subtree do not ripple to sibling subtrees.

We will explore using replication in space to improve the robustness of the system to node failures. Our main focus will be on developing a *supernode* [57] based approach in which k nodes join together to simulate a single virtual node and thereby mask temporary failures of individual nodes. A key challenge to using supernodes in an SDIMS system is developing a cooperative reliable messaging layer that ensures that all k nodes that are simulating the same node see the same sequence of incoming messages. Other research issues include node selection (balancing the robustness of using widely distributed nodes against the network efficiency of using nearby nodes) and retaining flexibility by allowing different attributes to use different degrees of replication. In addition to our supernodes based strategy, we also will explore a simpler replication-in-space strategy of allowing applications to use the up/down propagation flexibility to mask node failures: a failure of a descendent can be temporarily masked by installing attributes so that nodes propagate values at least one extra layer up the tree, and a failure of an ancestor can be temporarily masked by propagating aggregate results at least one extra layer down the tree. It will be interesting to compare the effectiveness of this simple approach to the more powerful but more complex supernodes approach.

We will also use replication in time to improve robustness. In particular, no matter how effective the stability-improvement and space-replication strategies are in reducing the number of reconfigurations of SDIMS's internal state, occasional reconfigurations are inevitable (e.g., when a node first joins or permanently leaves the system.) Unfortunately, the amount of state per node increases with the number of attributes in the system, so for SDIMS to be a general middleware that supports large numbers of attributes, we must take steps to mask this state transfer time so that it does not interfere with availability. We plan to adapt our volume leases recovery algorithms [96], which provides a way to quickly notice when a client and server have become desynchronized and to incrementally resynchronize the state while continuing to answer queries about the state by reconstructing what is required to answer demand requests. In particular, all stored state is associated with an *epoch number* and the current epoch is incremented when communication between the client and server fails; state can be resynchronized by replaying the missing messages or by scanning through the state, validating synchronization, and updating the epoch number. Demand requests to state from a prior epoch trigger communication that updates the state and epoch. Key challenges to adapting this algorithm to our environment include extending it to handle hierarchies and to maintain different epochs for different subspaces of keys created by the DHT construction. Also, in order to support our goal of flexibility, the API should allow application probes to accept stale data or to trigger reaggregation if the staleness of data exceeds some bound.

4 Application Prototyping

In addition to developing an SDIMS prototype we will use this prototype to construct several significant applications. The purpose of this work is twofold. First, to demonstrate that SDIMS, in fact, is a useful abstraction that qualitatively simplifies the development of large-scale networked systems. Second, to stress-test SDIMS under real world applications in order to identify and fix places where the initial SDIMS design is lacking.

We plan to examine three applications, which we describe in the rest of this section. In addition, we plan to make our software available to other researchers and to respond to their feedback from their experiences.

4.1 Scalable control for PRACTI replication

In other work, we are beginning to develop a unified replication toolkit to unify the development of large scale data replication systems such as enterprise replication [75], web caching [83], web prefetching [51], edge servers [37], and personal file systems [95]. In order to unify these disparate applications, the system must support the PRACTI

properties: Partial Replication, Arbitrary Consistency, and Topology Independence [27]. Conversely, past replication systems [41, 45, 50, 62, 65, 75, 82, 97] have only been able to provide at most two of these three properties. In order to achieve its goals, this PRACTI replication toolkit is designed to carefully separate the control path from the data path: the data path allows any node to safely exchange information with any other node, and the control path decides which nodes should exchange information.

We propose to use SDIMS to construct a scalable, distributed control system for an enterprise-scale file system built over the unified replication toolkit's flexible data path. Our initial analysis of the application suggests than a general SDIMS middleware could potentially be extremely helpful for the development of this file system by providing a common framework for a broad range of coordination tasks that it requires. In particular, we envision that the system's control plane will use SDIMS to (1) locate nearby copies of data to satisfy demand misses, (2) monitor the health of the replicated servers, (3) allow nodes that share each subset of data to rendezvous and form a topology-aware spanning tree for propagating updates about each subset of data, (4) allow nodes that miss a sequence of updates to identify nearby nodes that have the needed updates, and (5) to support aggressive self-tuning prefetching [51, 90, 89, 91] by tracking the update rate and read rate of each object in the system. It thus appears to us that SDIMS may enable a novel file system that is considerably more sophisticated along a number of dimensions than any that has been built in the past; without a common, scalable, flexible, robust aggregation framework, it is not clear that it would be feasible to construct such a system.

This PRACTI file system also appears to be an excellent stress test of key aspects of the SDIMS system. In particular, our initial analysis has suggested four issues that will challenge our SDIMS design. First, the file system may pose unexpected query types to the SDIMS system. For example, file-location has been a working example in much of our work to sketch an SDIMS design, but when we went from this abstract problem to the specific queries we would use in this file system, we realized that the file system case would be considerably more demanding than the abstract problem because the file system needs to be able to locate nearby copies of specific byte ranges within files rather than just locating nearby copies of whole files. Furthermore, because the PRACTI system does not constrain writes to any specific block size, the offsets in byte range queries can be arbitrary. In principle, we can define an aggregation function that uses the file ID as an attribute name and that maintains byterange data for each file, but it remains to be seen how complex programming such a function under our system will be. Also, note that the attribute value for each attribute name may be quite large—a sequence of $\{$offset, length, metadata$\}$ tuples—and that for efficiency it may be desirable to incrementally update subranges of the attribute value rather than always resending and recomputing the full attribute value whenever it changes. A second challenge posed by scalable PRACTI control to SDIMS is the scale of the number of attributes; our goal was for SDIMS to scale to millions of attribute values, and using SDIMS to track per-file control data for an enterprise-scale file system will test that aspect of our system. One key question is whether we will need to extend SDIMS to page data to disk to scale to a sufficient size. A third challenge is flexibility; attributes range considerably in read frequency, write frequency, global popularity, and latency sensitivity, so different propagation strategies will be required by different attribute types and by different attribute names at different times and places within the system. A fourth challenge is reliability: our file system will rely on SDIMS for correctness and performance, and our proposed methods for providing high availability and reliability for SDIMS will certainly be tested by this scalable replication application.

4.2 Grid information systems and scheduling

We plan to integrate the SDIMS implementation with the Globus Global Information Service (GIS) interface [20, 2] in order to significantly improve the state of the art and capabilities of GIS and the grid services and applications that depend on it. In particular, we believe that SDIMS's use of self-tuning peer-to-peer aggregation can provide a significant boost in capabilities compared to more traditional client-server architectures by scaling to more nodes, simplifying the deployment of distributed information managing services, supporting broader range of applications than is currently feasible, and simplifying the construction of applications that use the service.

A number of existing grid information systems such as GIS [2], MDS-2 [20], R-GMA [12], and Hawkeye [44] each provide a core distributed information management system designed to support a range of applications and services such as scheduling, replica selection, service discovery, and system monitoring. All of these systems use a client-server model in which *Information Providers* collect or generate data and supply this data to *Information Servers* through which applications access data. In some of these systems, scalability beyond this basic client-server model is provided by time to live (TTL) based caching, pushing of updates, and arrangement of Information Servers into cache hierarchies. However, Zhang et al. [98] find that existing systems have difficulty scaling to more than about 100 Information Providers and that scalability depends on placement of key components on well-connected and

well-provisioned machines.

We plan to study using SDIMS to improve on the implementation of the grid information services abstraction. In particular, Globus's GIS and MDS-2 define a common API, data model, and schema for grid information systems that is orthogonal to their underlying implementation [20, 30]. By extending SDIMS to support this interface, we can provide an enhanced implementation that is immediately usable by existing GIS or MDS-2 clients.

This work will test the hypothesis that an SDIMS-based GIS will provide several key advantages over existing GIS implementations.

- **Simplified deployment and scalability to large numbers of nodes.** SDIMS underlying DHT technology provides a framework for highly-effective self-tuning load balancing and locality [5, 40, 43, 49, 53, 58, 59, 61, 68, 70, 74, 79, 100]. In contrast, existing client-server architectures require configuration of specific well-connected and well-provisioned machines to act as Information Collectors and Information Servers for specific sets of machines, and they may require manual configuration of caching hierarchies for additional scalability.
- **Support for a broader range of applications and services.** The scalable forest of trees provided by SDIMS's underlying DHT framework allows the system to support applications such as global file location [24, 69, 72, 83] that tracks millions of attributes (fileIds) and to provide locally-relevant views (e.g., "find the nearest copy of */foo/bar*"), and that Although some researchers have proposed using GIS for a file-location service [20] it is not clear how to make such an approach scale to millions of files partially replicated across thousands of sites without configuring different cache hierarchies for each collection of data. As a result, data-location applications typically construct their own distributed information service. For example, the Storage Resource Broker (SRB) [6], a widely used grid data manager, uses a logically centralized metadata catalog.
- **Improved performance for applications and services.** The self-tuning replication we propose to develop for SDIMS will complement the DHT-based load balancing to improve the trade-offs among bandwidth, query latency, and consistency compared to existing approaches. Furthermore, by being self-tuning, gaining these advantages will be simpler than with manually-tuned systems.
- **Simplified development of applications and services.** SDIMS performs "in the network", "multi-scale" aggregation so that it is natural to query aggregate values at individual machines, clusters or collections of workstations on a subnet, the machines in a department, the machines in a virtual organization (VO) [33], or the machines "nearby" on the network. This uniformity allows a simple programming model where the programmer focuses on the aggregation function itself and the underlying system performs this aggregation at varying degrees of granularity, balances load across machines, and manages replication to balance overhead and performance. In contrast, MDS-2 is architected so that the "natural" level of aggregation is a cluster: Information Providers typically collect information from a pre-defined cluster of machines and Information Servers collect information from Information Providers and supply that information to a pre-defined cluster of client machines. Beyond the configuration and scalability challenges discussed above, this approach increases the complexity of doing aggregation at different levels of granularity and requires programming of separate abstractions for collecting and aggregating data within a hierarchy (at the Information Providers) and across hierarchies (at the Information Servers.)

We plan to proceed in three phases. First, we will develop techniques to integrate SDIMS with GIS or MDS-2. This work will include providing the GIS/MDS-2 API over the SDIMS system; our initial evaluation suggests that the abstractions are similar enough that this should be feasible. It will also include developing an internal interface for data transfer between SDIMS-based and traditional GIS implementations as well as enhancements of SDIMS to enforce the GIS security model. Second, once SDIMS and GIS share a common interface, we will be in a position to compare them directly and test the hypotheses discussed above. Third, we will apply this system to an effort by the Texas Advanced Computing Center (TACC) to develop and deploy a Community Scheduler Framework (CSF) [77] based scheduling framework for the University of Texas grid. An attached letter of support from Dr. John R. Boisseau, the Director of TACC, discusses his support for this collaboration.

4.3 Network monitoring

An emerging trend in network router design is to construct them using programmable network processing units such as AMCCs np7xxx, Ageres PaylodPlus, IBMs PowerNP, Silicon Accesss iFlow, Motorolas CPort, and Intels IXP. This trend will enable the deployment of sophisticated functionality in the network fabric such as intrusion detection, protocol conversion, QoS provisioning, XML firewalls, and VPNs. However high data rates and complex node architectures and sophisticated distributed algorithms makes programming such systems difficult. We will explore using SDIMS to address the distributed algorithms aspect of network monitoring systems.

We plan to examine the "distributed heavy hitter problem." The goal of the heavy hitter problem is to identify network sources, destinations, or protocols that account for significant or unusual amounts of traffic. As noted by Estan et al. [29], this information is useful for a variety of applications such as intrusion detection (e.g., port scanning), denial of service detection, worm detection and tracking, fair network allocation, or network maintenance. Significant work has been done on developing high-performance stream-processing algorithms for identifying heavy hitters at one router [29, 28, 31], but this is just a first step: ideally these applications would like not just one router's views of the heavy hitters but an aggregate view. We plan to use SDIMS to allow local information about heavy hitters to be pooled into a view of global heavy hitters. Initially, we will deploy this system as a distributed denial of service detector for PlanetLab [66] experiments (to flag experiments that accidently exceed acceptable loads on nodes outside of the PlanetLab testbed). We will then extend this work to developing a more general distributed heavy hitter monitoring system that runs on the Shangri-La runtime system [92] for programmable network routers.

We anticipate that this application class will stress a number of aspects of the SDIMS system. First, these network monitoring applications can generate large amounts of data on each node, so a challenge will be developing update propagation strategies that flexibly trade bandwidth for accuracy and then achieving an acceptable level of accuracy for an acceptable amount of bandwidth. Second, many of these network monitoring applications are security-related, so they are likely to drive us to further harden our design to secure the system from unauthorized inputs and to make the calculation of aggregate values robust against a small number of misbehaving nodes.

5 Conclusion

The goal of the proposed research is to design and build a Scalable Distributed Information Management System (SDIMS) that *aggregates* information about large-scale networked systems and that can serve as a basic building block for a broad range of large-scale distributed applications. Monitoring, querying, and reacting to changes in the state of a distributed system are core components of applications such as system management [11, 36, 71, 80, 84, 93], service placement [35, 94], grid scheduling [4, 7, 8, 17, 21, 47, 34, 32, 54, 76], data sharing and caching [61, 70, 74, 79, 83, 100], sensor monitoring and control [48, 56], multicast tree formation [14, 15, 86, 78, 81], and naming and request routing [16, 19]. We therefore speculate that an SDIMS in a networked system would provide a "distributed operating systems backbone" and facilitate the development and deployment of new distributed services.

References

[1] Keno Albrecht, Ruedi Arnold, Michael Gahwiler, and Roger Wattenhofer. Join and Leave in Peer-to-Peer Systems: The DASIS approach. Technical report, CS, ETH Zurich, 2003.

[2] "Globus Alliance". Ws information services: Key concepts. `ttp://www-unix.globus.org/toolkit/docs/3.2/infosvcs/ws/key/index.`tml, 2004.

[3] D. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the Eightteenth ACM Symposium on Operating Systems Principles*, pages 131–145. ACM Press, 2001.

[4] D. Angulo, I. Foster, C. Liu, and L. Yang. Design and evaluation of a resource selection framework for grid applications. In *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, July 2002.

[5] James Aspnes and Gauri Shah. Skip Graphs. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms*, January 2003.

[6] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC storage resource broker. In *CASCON98*, December 1998.

[7] Jim Basney and Miron Livny. Improving goodput by co-scheduling cpu and network capacity. *International Journal of High Performance Computing Applications*, 13(3), 1999.

[8] Jim Basney and Miron Livny. Managing network resources in Condor. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 298–299, Pittsburgh, PA, August 2000.

[9] Ranjita Bhagwan, Priya Mahadevan, George Varghese, and Geoff M. Voelker. Cone: A Distributed Heap-Based Approach to Resource Selection. Technical Report CS2004-0784, UCSD, 2004.

[10] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Comm. of the ACM*, 13(7):422–425, 1970.

[11] Rajkumar Buyya. PARMON: a portable and scalable monitoring system for clusters. *Software — Practice and Experience*, 30(7):723–739, 2000.

[12] R. Byrom, B. Coghlan, A. Cooke, R. Cordenonsi, L. Cornwall, A. Datta, A. Djaoui, L. Field, S. Fisher, S. Hicks, S. Kenny, J. Magowan, W. Nutt1, D. OCallaghan, M. Oevers, N. Podhorszki6, J. Ryan, M. Soni, P. Taylor, A. Wilson, and X. Zhu. R-GMA: A relational grid information and monitoring system. In *2nd Cracow Grid Workshop*, December 2002.

[13] M. Castro, Peter Druschel, Y. C. Hu, and A. Rowstron. Exploiting Network Proximity in Peer-to-Peer Overlay Networks. Technical Report MSR-TR-2002-82, MSR.

[14] M. Castro, Peter Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Multicast in a Cooperative Environment. In *SOSP*, 2003.

[15] M. Castro, Peter Druschel, A-M. Kermarrec, and A. Rowstron. SCRIBE: A Large-scale and Decentralised Application-level Multicast Infrastructure. *IEEE JSAC (Special issue on Network Support for Multicast Communications)*, 2002.

[16] Jim Challenger, Paul Dantzig, and Arun Iyengar. A scalable and highly available system for serving dynamic data at frequently accessed web sites. In *In Proceedings of ACM/IEEE, Supercomputing '98 (SC98)*, November 1998.

[17] S. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. The legion resource management system. In *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99)*, April 1999.

[18] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2002.

[19] Russ Cox, Athicha Muthitacharoen, and Robert T. Morris. Serving DNS using a Peer-to-Peer Lookup Service. In *IPTPS*, 2002.

[20] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, Aug 2001.

[21] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. *Lecture Notes in Computer Science 2537*, pages 153–183, 2002.

[22] F. Dabek, B. Zhao, P. Druschel, and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *2nd International Workshop on Peer-to-Peer Systems*, February 2003.

[23] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area Cooperative Storage with CFS. In *SOSP*, 2001.

[24] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001.

[25] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *IPTPS*, February 2003.

[26] M. Dahlin, B. Chandra, L. Gao, and A. Nayate. End-to-end WAN service availability. *ACM/IEEE Transactions on Networking*, 11(2), April 2003.

[27] M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula, and J. Zheng. PRACTI replication for large-scale systems. In review., May 2004.

[28] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *ACM SIGCOMM 2003 Conference*, 2003.

[29] C. Estan, G. Varghese, and M. Fisk. Bitmap algorithms for counting active flows on high speed links. In *Internet Measurement Conference 2003*, 2003.

[30] Steve Fitzgerald, Ian Foster, Carl Kesselman, Gregor von Laszewski, Warren Smith, and Steve Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, 5-8 August 1997.

[31] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences,*, 31(2):182–209, October 1985.

[32] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Intl Workshop on Quality of Service*, 1999.

[33] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.

[34] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, August 2001.

[35] Y. Fu, J. Chase, B. Chun, S. Schwab, and Amin Vahdat. SHARP: An architecture for secure resource peering. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, October 2003.

[36] Ganglia: Distributed Monitoring and Execution System. `http://ganglia.sourceforge.net`.

[37] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar. Application specific data replication for edge services. In *International World Wide Web Conference*, May 2003.

[38] Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, and Dan Suciu. What Can Peer-to-Peer Do for Databases, and Vice Versa? In *Proceedings of the WebDB*, 2001.

[39] Krishna Gummadi, Ramakrishna Gummadi, Steven D. Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *SIGCOMM*, 2003.

[40] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead. In *Proceedings of the 2nd International Workshop on Peer To Peer Systems (IPTPS)*, 2003.

[41] R. Guy, J. Heidemann, W. Mak, T. Page, Gerald J. Popek, and D. Rothmeier. Implementation of the Ficus Replicated File System. In *Proceedings of the Summer 1990 USENIX Conference*, pages 63–71, June 1990.

[42] B. Hardekopf, T. Riche, J. Mudigonda, M. Dahlin, H.M. Vin, and J. Kaur. Impact of network protocols on programmable router architectures, Apr 2003. In review.

[43] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *USITS*, March 2003.

[44] Hawkeye: A monitoring and management tool for distributed systems. http://www.cs.wisc.edu/condor/hawkeye/.

[45] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[46] Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the Internet with PIER. In *Proceedings of the VLDB Conference*, May 2003.

[47] IBM. *LoadLeveler for AIX 5L V3.2 Using and Administering*, October 2003. Document SA22-7881-01.

[48] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom*, 2000.

[49] Frans Kaashoek and David R. Karger. Koorde: A Simple Degree-Optimal Hash Table. In *Proceedings of the 2nd International Workshop on Peer To Peer Systems (IPTPS)*, 2003.

[50] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[51] R. Kokku, P. Yalagandula, A. Venkataramani, and M. Dahlin. A non-interfering deployable web prefetching system. In *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems*, March 2003.

[52] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the 9th ASPLOS*, November 2000.

[53] Xiaozhou Li and C. Greg Plaxton. On Name Resolution in Peer-to-Peer Networks. In *Proceedings of the POMC*, October 2002.

[54] Miron Livny and Rajesh Raman. High-throughput resource management. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[55] N. Lynch, D. Malkhi, and D. Ratajczak. Atomic data access in distributed hash tables. In *First International Workshop on Peer-to-Peer Computing*, number 2429 in Lecture Notes in Computer Science, pages 295–305. Springer, 2002.

[56] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *OSDI*, 2002.

[57] Dahlia Malkhi. Dynamic Lookup Networks. In *FuDiCo*, 2002.

[58] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC)*, 2002.

[59] Gurmeet Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed Hashing in a Small World. In *Proceedings of the USITS conference*, 2003.

[60] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. In submission.

[61] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *Proceesings of the IPTPS*, March 2002.

[62] M. Nelson, B. Welch, and J. Ousterhout. Caching in the Sprite Network File System. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[63] C. Olston, B. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *SIGMOD Intl. Conference on Management of Data*, pages 355–366, May 2001.

[64] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Twenty-Sixth International Conference on Very Large Data Bases*, pages 144–155, September 2000.

[65] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and A. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, October 1997.

[66] L. Peterson, D. Culler, and T. Anderson. Planetlab: A testbed for developing and deploying network services. http://www.planet-lab.org/pubs/vision.pdf, June 2002.

[67] Planetlab. `http://www.planet-lab.org`.

[68] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In *ACM SPAA*, 1997.

[69] G. Plaxton, R. Rajaram, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320, June 1997.

[70] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM*, 2001.

[71] Timothy Roscoe, Richard Mortier, Paul Jardetzky, and Steven Hand. InfoSpect: Using a Logic Language for System Health Monitoring in Distributed Systems. In *Proceedings of the SIGOPS European Workshop*, 2002.

[72] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eightteenth ACM Symposium on Operating Systems Principles*, 2001.

[73] A. Rowstron and Peter Druschel. Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility. In *SOSP01*, October 2001.

[74] Antony Rowstron and Peter Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems. In *Middleware*, 2001.

[75] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the pangaea wide-area file system. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

[76] V. Sander, W. A. Adamson, I. Foster, and A. Roy. End-to-end provision of policy information for network QoS. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC-10*, August 2001.

[77] C. Smith. Open source metascheduling for virtual organizations with the community scheduler framework (CSF). Technical whitepaper, Platform Computing, 2003.

[78] S.Ratnasamy, M.Handley, R.Karp, and S.Shenker. Application-level Multicast using Content-addressable Networks. In *Proceedings of the NGC*, November 2001.

[79] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *ACM SIGCOMM*, 2001.

[80] Supermon: High speed cluster monitoring. `http://www.acl.lanl.gov/supermon/`.

[81] S.Zhuang, B.Zhao, A.Joseph, R.Katz, and J.Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *NOSSDAV*, 2001.

[82] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the Fifteenth ACMSymposium on Operating Systems Principles*, pages 172–183, December 1995.

[83] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In *Proceedings of the Nineteenth International Conference on Distributed Computing Systems*, May 1999.

[84] IBM Tivoli Monitoring. `www.ibm.com/software/tivoli/products/monitor`.

[85] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.

[86] Robbert VanRenesse, Kenneth P Birman, and Werner Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *TOCS*, 2003.

[87] Robbert VanRenesse and Adrian Bozdog. Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol. In *IPTPS*, 2004.

[88] A. Venkataramani, R. Kokku, and M. Dahlin. Operating system support for massive replication. In *Proceedings of the 10th ACM SIGOPS European Workshop*, September 2002.

[89] A. Venkataramani, R. Kokku, and M. Dahlin. TCP-Nice: A mechanism for background transfers. In *OSDI02*, December 2002.

[90] A. Venkataramani, P. Weidmann, and M. Dahlin. Bandwidth constrained placement in a wan. In *Proceedings of the Twentieth Symposium on the Principles of Distributed Computing*, August 2001.

[91] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. Potential costs and benefits of long-term prefetching for content-distribution. *Elsevier Computer Communications*, 25(4):367–375, March 2002.

[92] H. Vin. Shgangri-la: A programming environment for next-generation network systems. http://www.cs.utexas.edu/users/vin/research/shangrila.shtml, 2003.

[93] Mike Wawrzoniak, Larry Peterson, and Timothy Roscoe. Sophia: An Information Plane for Networked Systems. In *HotNets-II*, 2003.

[94] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, Oct 1999.

[95] P. Yalagandula, L. Alvisi, M. Dahlin, and H. Vin. Consistent o-administration personal environment. In *IEEE 6 th International Workshop on Object Oriented Real-Time Dependable Systems*, January 2001.

[96] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering web cache consistency. *ACM Transactions on Internet Technologies*, 2(3):224–259, 2002.

[97] H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems*, 20(3), August 2002.

[98] X. Zhang, J. Freschl, and J. Schopf. A performance study of monitoring and information services for distributed systems. In *The 12th International Symposium on High Performance Distributed Computing*, August 2003.

[99] Zheng Zhang, Shu-Ming Shi, and Jing Zhu. SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT. In *IPTPS*, 2003.

[100] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.