2ND QUARTER 2008, VOLUME 10, NO. 2

IEEE COMMUNICATIONS SURVEYS

www.comsoc.org/pubs/surveys

PEER-TO-PEER-BASED RESOURCE DISCOVERY IN GLOBAL GRIDS: A TUTORIAL

RAJIV RANJAN, AARON HARWOOD AND RAJKUMAR BUYYA, THE UNIVERSITY OF MELBOURNE

ABSTRACT

An efficient resource discovery mechanism is one of the fundamental requirements for grid computing systems, as it aids in resource management and scheduling of applications. Resource discovery activity involves searching for the appropriate resource types that match the user's application requirements. Various kinds of solutions to grid resource discovery have been suggested, including centralized and hierarchical information server approaches. However, both of these approaches have serious limitations in regard to *scalability*, *fault tolerance*, *and network congestion*. To overcome these limitations, indexing resource information using a decentralized (e.g., peer-to-peer (P2P)) network model has been actively proposed in the past few years. This article investigates various decentralized resource discovery techniques primarily driven by the P2P network model. To summarize, this article presents a: summary of the current state of the art in grid resource discovery, resource taxonomy with focus on the computational grid paradigm, P2P taxonomy with a focus on extending the current structured systems (e.g., distributed hash tables) for indexing *d*-dimensional grid resource queries,[1] a detailed survey of existing work that can support *d*-dimensional grid resource queries, and classification of the surveyed approaches based on the proposed P2P taxonomy. We believe that this taxonomy and its mapping to relevant systems would be useful for academic and industry-based researchers who are engaged in the design of scalable grid and P2P systems.

The last few years have seen the emergence of a new generation of distributed systems that scale over the Internet, operate under decentralized settings, and are dynamic in their behavior (participants can leave or join the system). One such system is referred to as grid computing; other similar systems include peer-to-peer (P2P) computing [1], semantic Web [2], pervasive computing [3], and mobile computing [4, 5]. Grid computing [6] provides the basic infrastructure required for sharing diverse sets of resources, including desktops, computational clusters, supercomputers, storage, data, sensors, applications, and online scientific instruments. Grid computing offers its vast computational power to solve grand challenge problems in science and engineering such as protein folding, high energy physics, financial modeling, earthquake simulation, climate/weather modeling, aircraft engine diagnostics, earthquake engineering, virtual observatory, bioinformatics, drug discovery, digital image analysis, astrophysics, and multi-player gaming.

Grids can primarily be classified [7] into various types, depending on the nature of their emphasis: computation, data, application service, interaction, knowledge, and utility. Accordingly, grids are proposed as the emerging cyber infrastructure to power utility computing applications. Computational grids aggregate the computational power of globally distributed computers (e.g., TeraGrid, ChinaGrid, and APACGrid). Data grids emphasize global-scale management of data to provide data access, integration, and processing through distributed data repositories (e.g., LHCGrid, Gri-PhyN). Application service (provisioning) grids focus on providing access to remote applications and modules; libraries hosted on data centers or computational grids (e.g., NetSolve and GridSolve). Interaction grids focus on interaction and collaborative visualization between participants (e.g., AccessGrid). Knowledge grids aim toward knowledge acquisition, processing, and management, and provide business analytics services driven by integrated data mining services. Utility grids focus on providing all the grid services, including compute power, data, and service to end users as IT utilities on a subscription basis, and provide the infrastructure necessary for negotiation of required quality of service, establishment and management of contracts, and allocation of resources to meet competing demands. To summarize, these grids follow a layered design

[1] *The article assumes that the reader has a basic understanding of database indexing techniques.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

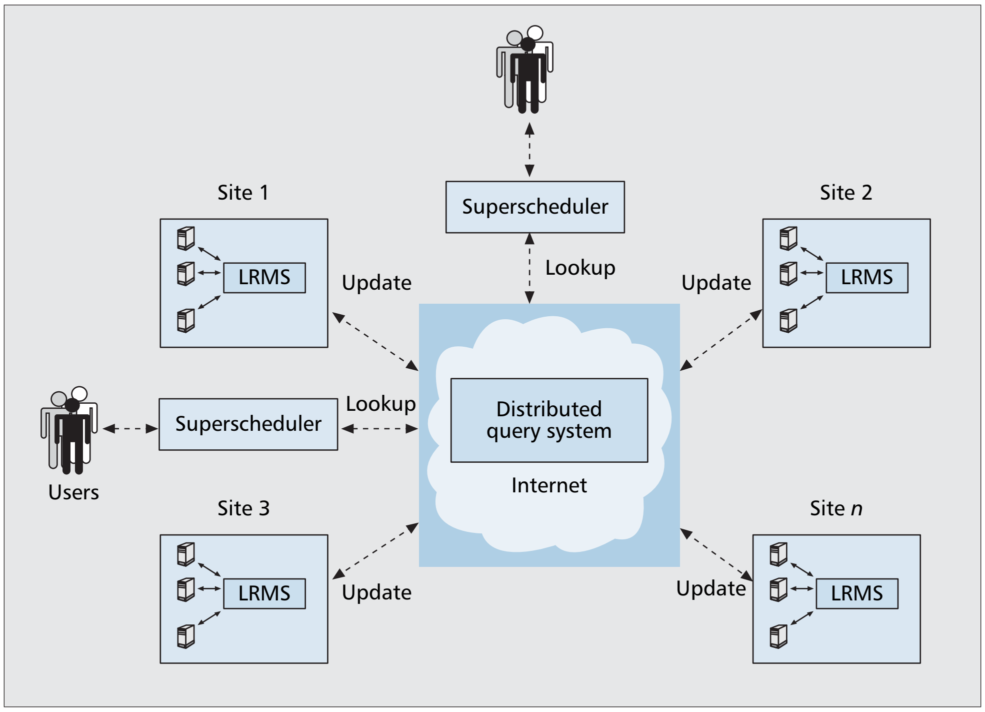


Figure 1. *Superscheduling and resource queries.*

with the computational grid being the bottom layer and the utility grid the top layer. A grid at a higher level utilizes the services of grids that operate at lower layers in the design. For example, a data grid utilizes the services of a computational grid for data processing and hence builds on it. In addition, lower-level grids focus heavily on infrastructure aspects, whereas higher-level ones focus on users and quality of service delivery.

In this work, we mainly focus on computational grids. Computational grids enable aggregation of different types of computing resources including clusters, supercomputers, and desktops. In general, computing resources have two types of attributes:

- Static attributes such as the type of operating system installed, network bandwidth (both local area network [LAN] and wide area network [WAN] interconnection), processor speed, and storage capacity (including physical and secondary memory);
- Dynamic attributes such as processor utilization, physical memory utilization, free secondary memory size, current usage price, and network bandwidth utilization.

The Superscheduling Process and Resource Indexing

The grid superscheduling [8] problem is defined as: "*scheduling jobs across the Grid resources such as computational clusters, parallel supercomputers, desktop machines that belong to different administrative domains*." Superscheduling in computational grids is facilitated by specialized grid schedulers/brokers such as the Grid Federation Agent [9], MyGrid [10], NASA-Superscheduler [11], Nimrod-G [12], GridBus-Broker [13], Condor-G [14], and workflow engines [15, 16]. Figure 1 shows an abstract model of a decentralized superscheduling system over a distributed query system. The superschedulers access the resource information by issuing lookup queries. The resource providers register the resource information through update queries. Superscheduling involves:

- Identifying and analyzing a user's job requirements;
- Querying a grid resource information service (GRIS) [17–22] for locating resources that match the job requirements;
- Coordinating and negotiating a service level agreement (SLA) [23–26];
- Job scheduling.

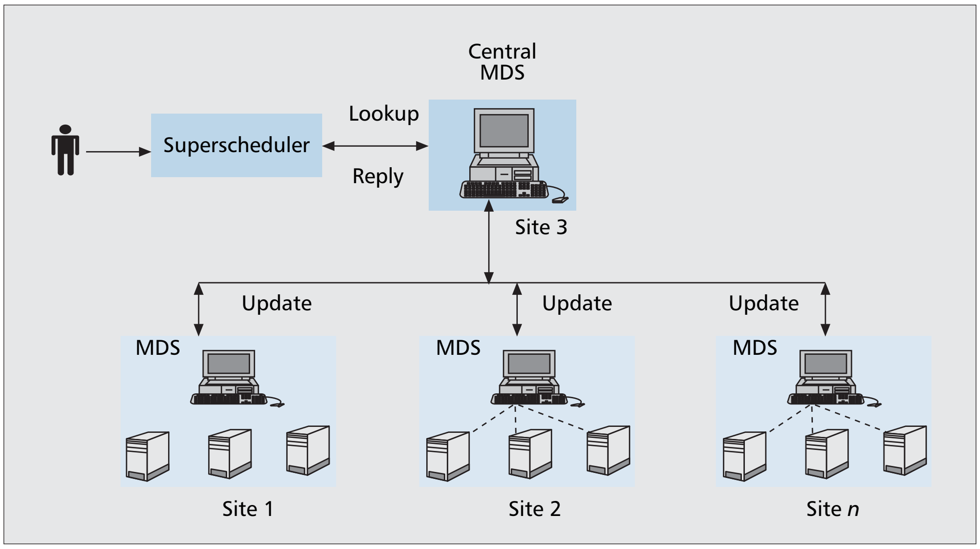
Grid resources are managed by their local resource management systems (LRMSs) such as Condor [27], Portable Batch System (PBS) [28], Sun Grid Engine (SGE) [29], Legion [30], Alchemi [31], and Load Sharing Facility (LSF) [32]. The LRMSs manage job queues, and initiate and monitor their execution.

Traditionally, superschedulers [33], including Nimrod-G, Condor-G, and Tycoon [34], used centralized information services such as R-GMA [35], Hawkeye [36], GMD [20], and MDS-1 [37] to index resource information. Under centralized organization, the superschedulers send resource queries to a centralized resource indexing service. Similarly, the resource providers update the resource status at periodic intervals using resource update messages. This approach has several design issues, including:

- Highly prone to a single point of failure;
- Lacks scalability;
- High network communication cost at links leading to the information server (i.e., network bottleneck, congestion);
- The machine running the information services might lack the required computational power required to serve a large number of resource queries and updates.

To overcome the above shortcomings of centralized approaches, a hierarchical organization of information services has been proposed in systems such as MDS-3 [19] and Ganglia [38]. MDS-3 organizes virtual organization (VO) [6] specific information directories in a hierarchy. A VO includes a set of GPs that agree on common resource sharing policies. Every VO in a grid designates a machine that hosts the information services. A similar approach has been followed in the Ganglia system, which is designed for monitoring resources status within a federation of clusters. Each cluster designates a node as a representative to the federated monitoring system. This node is responsible for reporting cluster status to the federation. However, this approach also has similar problems to the centralized approach such as one point of failure, and does not scale well for a large number of users/providers.

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

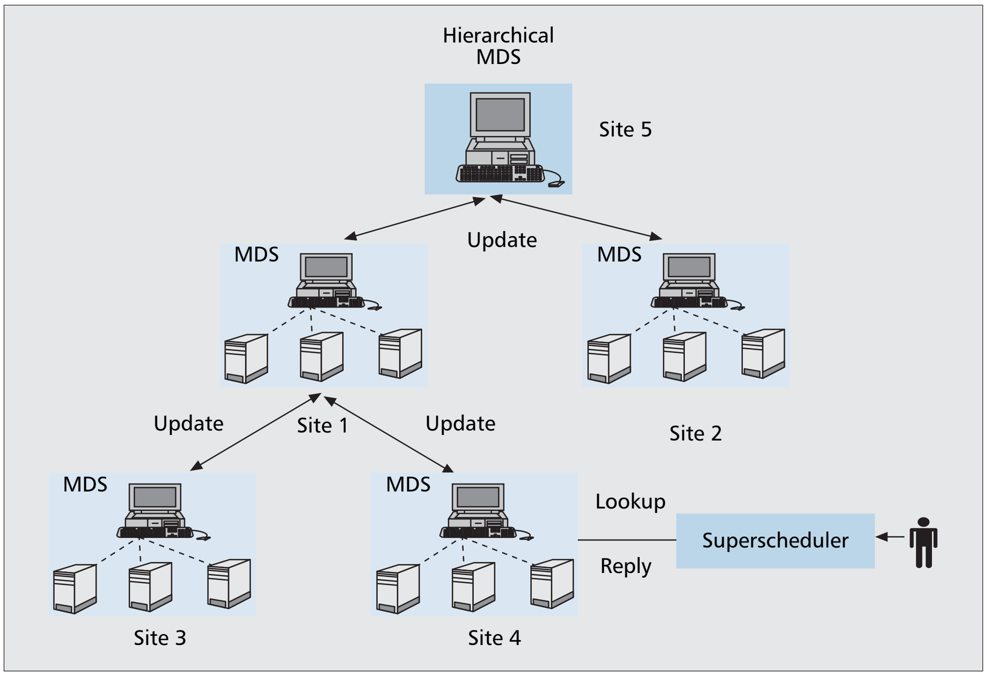


■ **Figure 2.** *A centralized monitoring and directory sservice (MDS) organization.*

Decentralized Resource Indexing

Recently, proposals for decentralizing a GRIS have gained significant momentum. The decentralization of GRIS can overcome the issues related to current centralized and hierarchical organizations. A distributed system configuration is considered as decentralized *if none of the participants in the system is more important than others, in case one of the participant fails, it is neither more or less harmful to the system than the failure of any other participant in the system*. An early proposal for decentralizing grid information services was made by Iamnitchi and Foster [18]. The work proposed a P2P-based approach for organizing the MDS directories in a flat dynamic P2P network. It envisages that every VO maintains its information services and makes it available as part of a P2P-based network. In other words, information services are the peers in a P2P network-based coupling of VOs. Application schedulers in various VOs initiate a resource lookup query that is forwarded in the P2P network using flooding (an approach similar to one applied in the unstructured P2P network Gnutella) [39]. However, this approach has a large volume of network messages generated due to flooding. To avoid this, a time to live (TTL) field is associated with every message (i.e., the peers stop forwarding a query message once the TTL expires). To an extent, this approach can limit the network message traffic, but the search query results may not be deterministic in all cases. Thus, the proposed approach cannot guarantee to find the desired resource even though it exists in the network.

Recently, organizing a GRIS over structured P2P networks has been widely explored. Structured P2P networks offer deterministic search query results with logarithmic bounds on network message complexity. Structured P2P lookup systems, including Chord [40], CAN [41], Pastry [42], and Tapestry [43], are primarily based on distributed hash tables (DHTs). DHTs provide hash-table-like functionality on the Internet scale. A DHT is a data structure that associates a key with data. Entries in the DHT are stored as a (key, data) pair.



■ **Figure 3.** *A hierarchical MDS organization.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

Level 0	Level 1	Level 2	Level 3
MapCenter [56], GridICE [57]	Autopilot [58]	CODE [59], GridRM [60], Hawkeye [36], HBM [61], Mercury [62], NetLogger [63], NWS [64], OCM-G [65], Remos [66], SCALEA-G [67]	Ganglia [38], Globus MDS [19], MonALISA [68], Paradyn [69], RGMA [35]

Table 1. *Summarizing centralized and hierarchical GRIS.*

Data can be looked up within a logarithmic overlay routing hop if the corresponding key is known.

It is widely accepted that DHTs are the building blocks for next-generation large-scale decentralized systems. Some of the example distributed systems that utilize the DHT routing substrate include distributed databases [44], group communication [45], email services [46], resource discovery systems [21, 47–50], and distributed storage systems [51]. Current implementations of DHTs are known to be efficient for one-dimensional queries [44] such as "find all resources that match the given search point." In this case distinct attribute values are specified for resource attributes. Extending DHTs to support *d*-dimensional range queries such as finding all resources that overlap a given search space is a complex problem. Range queries are based on range of values for attributes rather than a specific value. Current works including [17, 21, 22, 48–50, 52–55] have studied and proposed different solutions to this problem.

The State of the Art in Grid Information Indexing

The work in [36] presents a comprehensive taxonomy on existing centralized and hierarchically organized GRISs. Figure 2 shows centralized GRISs based on Globus MDS-2/3. A central server (at site 3) machine hosts the grid index information service (GIIS) (a component MDS-2), while the remaining grid sites run the GRIS. In Fig. 2 sites 1, 2, and *n* are running instances of GRIS that periodically update their resource status with the centralized GIIS. Figure 3 depicts a hierarchical information service organization using Globus MDS-2/3. In Fig. 3 sites 3 and 4 run the GRIS that connects to the GIIS hosted at site 1. Note that site 1 hosts both the GIIS and GRIS, and updates the information about its local resources along with child GRISs with the root GIIS service hosted at site 5.

We summarize this work here and classify existing systems according to the proposed taxonomy in Table 1. The proposed taxonomy is based on the grid monitoring architecture (GMA) [70] put forward by the Global Grid Forum (GGF). The main components of GMA are:
- Producer: a daemon that monitors and publishes resource attributes to the registry;
- Consumer: superschedulers that query the registry for resource information;
- Registry: a service or directory that allows publishing and indexing of resource information;
- Republisher: any object that implements both producer and consumer functionality;
- Schema repository: holds details such as type and schema about different kinds of events that are ambient in a GRIS.

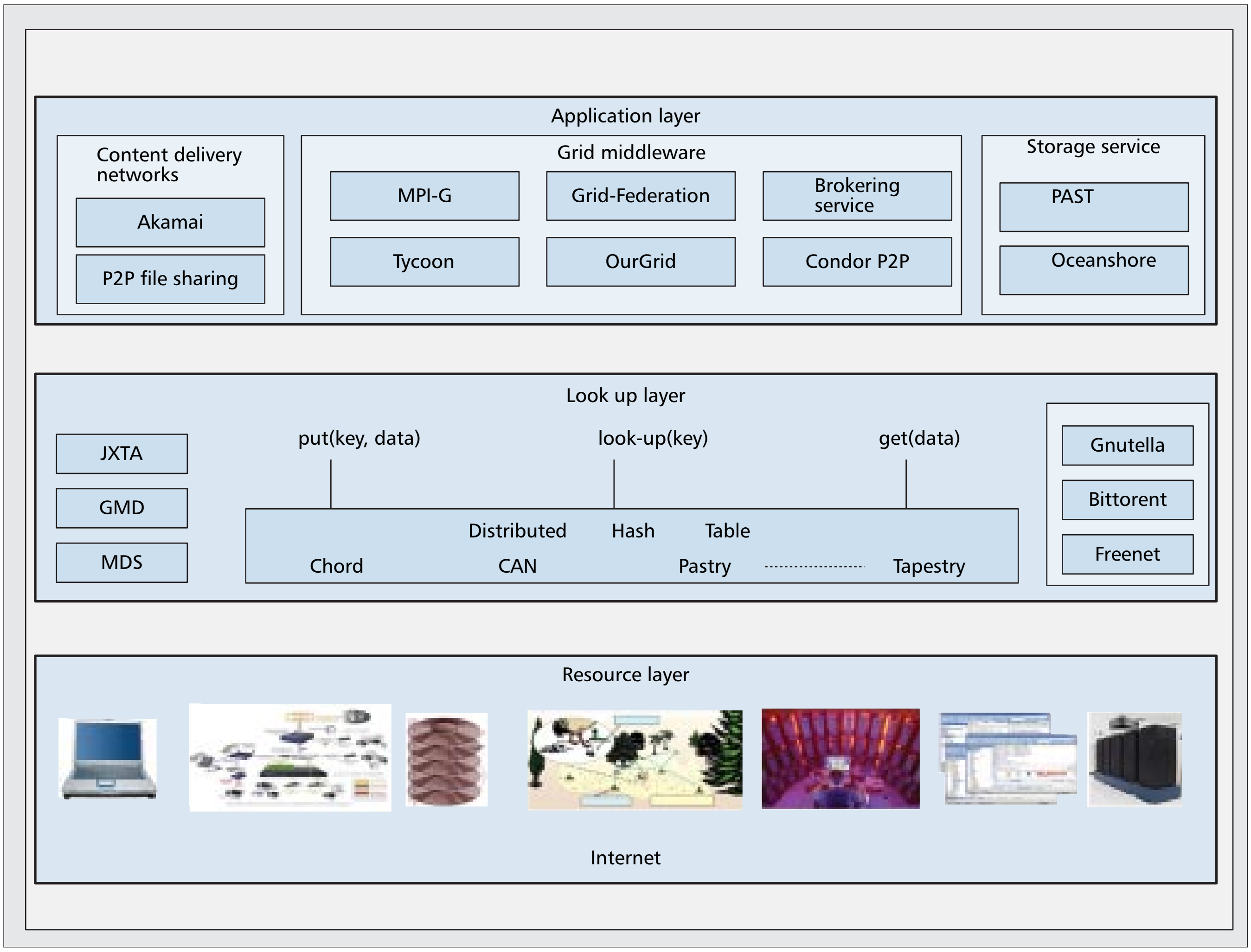
Systems are identified depending on the provision and characteristics of producers and republishers:
- Level 0 (self-contained systems): The resource consumers are directly informed of various resource attribute changes by the sensor daemon (a server program attached to the resource for monitoring its status). The notification process may take place in an offline or online setting. In the online case the sensors locally store the resource metrics, which can be accessed in an application-specific way. These systems normally offer a browsable Web interface that provides interactive access to HTML-formatted information. These systems do not provide any kind of producer application programming interface (API), thus lacking any programming support that can enable automatic distribution of events to remotely deployed applications. Systems including MapCenter [56] and GridICE [57] belong to level 0 resource monitoring systems.
- Level 1 (producer-only systems): Systems in this category have event sensors hosted on the same machine as the producer, or the sensor daemon functionality is provided by the producer itself. Additionally, these systems provide an API at the resource level (producer level); hence, they are easily and automatically accessible from remote applications. In this case there is no need to browse through the Web interface in order to gather resource information. Systems including Autopilot [58] belong to the level 1 category of monitoring systems.
- Level 2 (producers and republishers): This category of system includes a republisher attached to each producer. Republishers of different functionality may be stacked on each other but only in a predefined way. The only difference from level 1 systems is the presence of a republisher in the system. Systems including GridRM [60], CODE [59], and Hawkeye are level 2 systems.
- Level 3 (hierarchies of republishers): This category of system allows for the hierarchical organization of republishers in an arbitrary fashion. This functionality is not supported in level 2 systems. In this arrangement every node collects and processes events from its lower-level producers and republishers. These systems provide better scalability than a level 0, 1, or 2 system. Systems such as MDS-3 [19] belong to this category.

The taxonomy also proposes three other dimensions/qualifiers to characterize the existing systems. They include:
- Multiplicity: This qualifier refers to the scalability aspect (organization of the republisher in a level 2 system) of a GRIS. A republisher can be completely centralized or distributed with support of replication.
- Type of entities: Denotes types of resources indexed by a GRIS. Different resource types include hosts, networks, applications, and generic. A generic resource type at least supports event for hosts and network types.
- Stackable: Denotes whether the concerned GRIS can work on top of another GRIS.

Conceptual Design of a Distributed Resource Indexing System

A layered architecture to build a distributed resource indexing system is shown in Fig. 4. The key components of an Internet-based resource indexing system include:
- **Resource layer**: This layer consists of all globally distributed resources that are directly connected to the Internet. The range of resources include desktop machines, files, supercomputers, computational clusters, storage devices, databases, scientific instruments, and sensor networks. A computational resource can run variants of operating systems (e.g., UNIX or Windows) and queuing systems (e.g., Condor, Alchemi, SGE, PBS, LSF).
- **Lookup layer**: This layer offers core services for indexing resources on the Internet scale. The main components at

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.



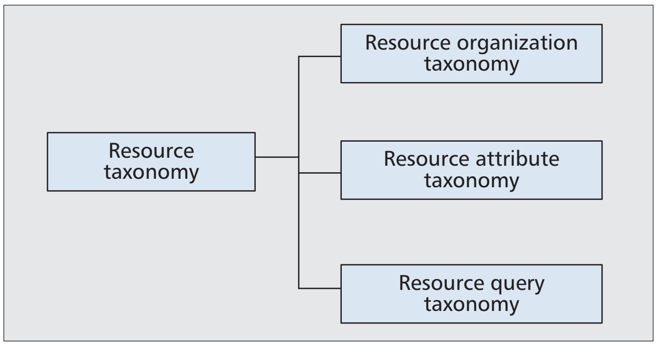
■ **Figure 4.** *Distributed resource indexing: a layered approach.*

this layer are the middleware that supports Internet-wide resource lookups. Recent proposals at this layer have been utilizing structured P2P protocols such as Chord, CAN, Pastry, and Tapestry. DHTs offer deterministic search query performance while guaranteeing logarithmic bounds on the network message complexity. Other, middlewares at this layer includes JXTA [71], Grid Market Directory (GMD), [20] and unstructured P2P substrates such as Gnutella [39] and Freenet [72].

- **Application layer**: This layer includes the application services in various domains, including grid computing, distributed storage, P2P networks, and content delivery networks (CDNs) [73, 74]. Grid computing systems, including Condor-Flock P2P [75], use services of Pastry DHT to index condor pools distributed over the Internet. Grid brokering system such as the Nimrod-G utilizes directory services of Globus [76] for resource indexing and superscheduling. The OurGrid superscheduling framework incorporates JXTA for enabling communication between OGPeers in the network. Distributed storage systems including PAST [77] and OceanStore [78] utilize services of DHTs such as Pastry and Tapestry for resource indexing.

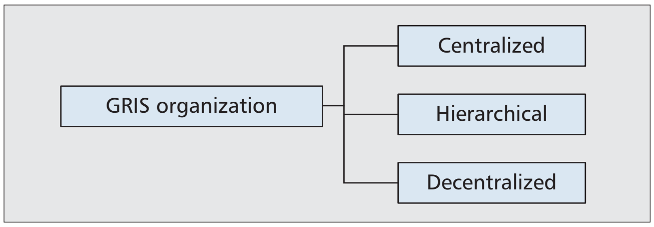
ARTICLE ORGANIZATION

The rest of the article is organized as follows. First, we present taxonomies related to general computational resources' attributes, lookup queries, and organization model. Next, we present taxonomies for P2P network organization, a *d*-dimensional data distribution mechanism, and a query routing mechanism. Then we summarize various algorithms that model GRIS over a P2P network, and compare the surveyed algorithms based on their scalability and index load balancing capability. We discuss current grid and P2P systems' security approaches and their limitations, and provide a recommendation on utilizing the surveyed approaches in implementing a resource discovery system. Finally, we end this article with a discussion of open issues and a conclusion.



■ **Figure 5.** *Resource taxonomy.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.



■ **Figure 6.** *Resource organization taxonomy.*

RESOURCE TAXONOMY

The taxonomy for a computational grid resource is divided into the following (Fig. 5): resource organization, resource attribute, and resource query.

RESOURCE/GRIS ORGANIZATION TAXONOMY

The taxonomy defines GRIS organization as (Fig. 6):

- Centralized: Centralization refers to the allocation of all query processing capability to a single resource. The main characteristics of a centralized approach include control and efficiency. All lookup and update queries are sent to a single entity in the system. GRISs including RGMA [35] and GMD [20] are based on centralized organization.
- Hierarchical: A hierarchical approach links GRISs either directly or indirectly, and either vertically or horizontally. The only direct links in a hierarchy are from the parent nodes to their child nodes. A hierarchy usually forms a tree-like structure. GRIS systems including MDS-3 [19] and Ganglia [38] are based on this network model.
- Decentralized: No centralized control; complete autonomy, authority, and query processing capability are distributed over all resources in the system. The GRIS organized under this model is fault-tolerant, self-organizing, and scalable to a large number of resources. More details on this organization can be found later.

There are four fundamental challenges related to different organization models: scalability, adaptability, availability, and manageability. Centralized models are easy to manage but do not scale well. When network links leading to the central server get congested or fail, performance suffers. Hence, this approach may not adapt well to dynamic network conditions. Furthermore, it presents a single point of failure, so the overall availability of the system degrades considerably. Hierarchical organization overcomes some of these limitations, including scalability, adaptability, and availability. However, these advantages over a centralized model come at the cost of overall system manageability. In this case every site-specific administrator has to periodically ensure the functionality of their local daemons. Furthermore, the root node in the system may present a single point of failure similar to the centralized model. Decentralized systems, including P2P, are highly scalable, adaptable to network conditions, and highly available. But manageability is a complex task in P2P networks as it incurs a lot of network traffic.

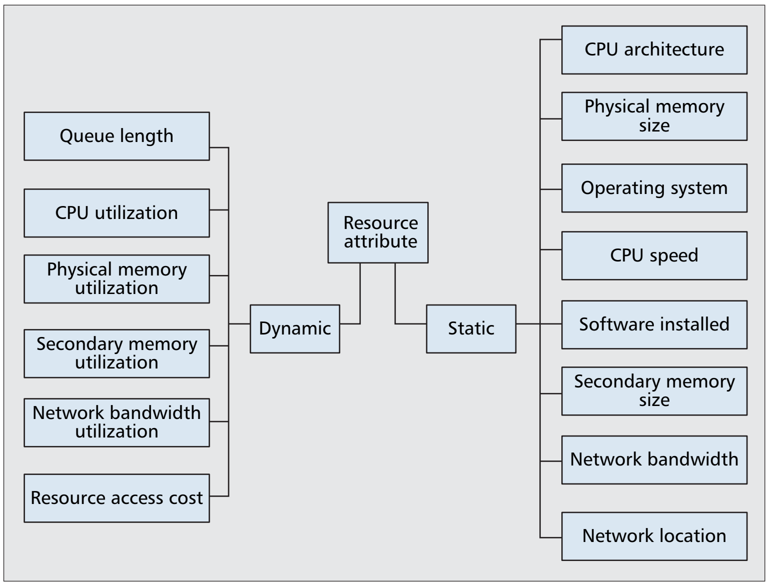
RESOURCE ATTRIBUTE TAXONOMY

A compute grid resource is described by a set of attributes that is globally known to the application superschedulers. A superscheduler interested in finding a resource to execute a user's job issues queries to GRIS. The queries are a combination of desired attribute values or their ranges, depending on the user's job composition. In general, computing resources have two types of attributes, static and fixed value attributes, such as type of operating system installed, network bandwidth (both LAN and WAN interconnection), network location, CPU speed, CPU architecture, software library installed, and storage capacity (including physical and secondary memory); and dynamic or range valued attributes such as CPU utilization, physical memory utilization, free secondary memory size, current usage price, and network bandwidth utilization. Figure 7 depicts the resource attribute taxonomy.

RESOURCE QUERY TAXONOMY

The ability of superschedulers such as MyGrid, Grid-Federation Agent, NASA-Superscheduler, Nimrod-G, and Condor-Flock P2P to make effective application scheduling decisions is directly governed by the efficiency of GRIS. Superschedulers need to query a GRIS to compile information about resource's utilization, load, and current access price for formulating efficient schedules. Furthermore, a superscheduler can also query a GRIS for resources based on selected attributes such as nodes with large amounts of physical and secondary memory, inter-resource attributes such as network latency, number of routing hops, or physical attributes such as geographic location. Similarly, the resource owners query a GRIS to determine supply and demand patterns and set the price accordingly. The actual semantics of the resource query depends on the underlying grid superscheduling model or grid system model.

Resource Query Type — Superscheduling systems require two basic types of queries: resource lookup query (RLQ) and resource update query (RUQ). An RLQ is issued by a superscheduler to locate resources matching a user's job requirements, while an RUQ is an update message sent to a GRIS by a resource owner about the underlying resource conditions. In the Condor-Flock P2P system, flocking requires sending RLQs to remote pools for resource status and the willingness to accept remote jobs. Willingness to accept remote jobs is a policy-specific issue. After receiving an RLQ message, the contacted pool manager replies with an RUQ that includes the job queue length, average pool utilization, and number of resources available. Distributed flocking is based on the P2P



■ **Figure 7.** *Resource attribute taxonomy.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

System name	Resource lookup query	Resource update qquery	GRIS model
Condor-Flock P2P	Query remote pools in the routing table for resource status and resource sharing policy	Queue length, average pool utilization and number of resources available	Decentralized
Grid-Federation	Query decentralized federation directory for resources that matches user's job QoS requirement (CPU architecture, no. of processors, available memory, CPU speed)	Update resource access price and resource conditions (CPU utilization, memory, disk space, no. of free processors)	Decentralized
Nimrod-G	Query GMD or MDS for resources that matches jobs resource and QoS requirement	Update resource service price and resource type available	Centralized
Condor-G	Query for available resource using Grid Resource Information Protocol (GRIP), then individual resources are queried for current status depending on superscheduling method	Update resource information to MDS using GRRP	Centralized
Our-Grid	MyPeer queries OGPeer for resources that match user's job requirements	Update network of favors credit for Our-Grid sites in the community	Decentralized
Gridbus Broker	Query GMD or MDS for resources that matches jobs resource and QoS requirement	Update resource service price and resource type available	Centralized
Tycoon	Query for auctioneers that are currently accepting bids and matches user's resource requirement	Update number of bids currently active and current resource availability condition	Centralized
Bellagio	Query for resources based on CPU load, available memory, internode latency, physical and logical proximity	Update resource conditions including CPPU , memory and network usage status	Decentralized
Mosix-Grid	Information available at each node through gossiping algorithm	Update CPU usage, current load, memory status and network status	Hierarchical

Table 2. *Resource query in superscheduling systems.*

query mechanism. Once the job is migrated to the remote pool, a basic matchmaking [79] mechanism is applied for resource allocation. In Table 2 we present RLQ and RUQ queries in some well-known superscheduling systems.

An Example Superscheduling Resource Query — In this section we briefly analyze the superscheduling query composition in the superscheduling system called Tycoon [34]. The Tycoon system applies market-based principles, in particular an auction mechanism, for resource management. Auctions are completely independent without any centralized control. Every resource owner in the system coordinates its own auction for local resources. The Tycoon system provides a centralized service location service (SLS) for superschedulers to index resource auctioneers' information. Auctioneers register their status with the SLS every 30 s. If an auctioneer fails to update its information within 120 s, the SLS deletes its entry. Application-level superschedulers contact the SLS to gather information about various auctioneers in the system. Once this information is available, the superschedulers (on behalf of users) issue bids for different resources (controlled by different auctions), constrained by resource requirements and available budget. A resource bid is defined by the tuple (h, r, b, t) where h is the host to bid on, r is the resource type, b is the number of credits to bid, and t is the time interval over which to bid. Auctioneers determine the outcome by using a bid-based proportional resource sharing economy model.

Auctioneers in the Tycoon superscheduling system send an RUQ to the centralized GRIS (referred to as SLS). The update message consists of the total number of bids currently active for each resource type and the total amount of each resource type available (e.g., CPU speed, memory size, disk space). An auctioneer's RUQ has the following semantics:

total bids = 10 && *CPU Arch* = "*pentium*" && *CPU Speed* = 2 *GHz* && *Memory* = 512

Similarly, the superscheduler, on behalf of the Tycoon users, issues an RLQ to the GRIS to acquire information about active resource auctioneers in the system. A user resource lookup query has the following semantics:

return auctioneers whose CPU Arch = "*i686*" && *CPU Speed* ≥ 1 *GHz* && *Memory* ≥ 256

In Fig. 8 we present the taxonomy for GRIS RLQ and RUQ. In general, the queries [80] can be abstracted as lookups for objects based on a single dimension or multiple dimensions. Since a grid resource is identified by more than one attribute, an RLQ or RUQ is always d-dimensional. Furthermore, both the one-dimensional and d-dimensional queries can specify different kinds of constraints on the attribute values. If the query specifies a fixed value for each attribute, it is referred to as a d-dimensional point query

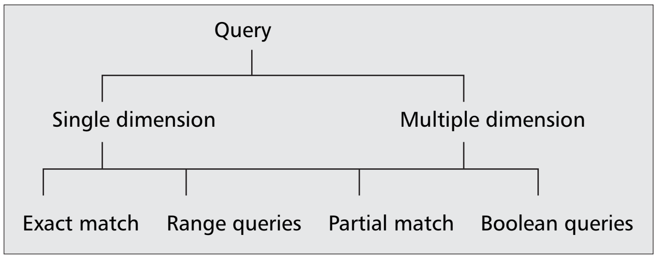
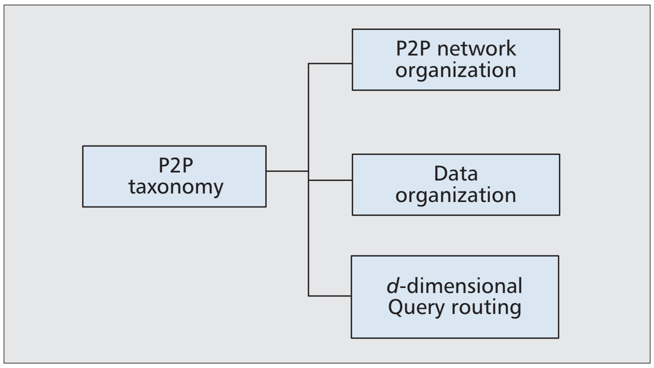


Figure 8. *Resource query taxonomy.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.



■ **Figure 9.** *Peer-to-peer network taxonomy.*

(DPQ). However, if the query specifies a range of values for attributes, it is referred to as a *d*-dimensional window query (DWQ) or *d*-dimensional range query (DRQ). Depending on how values are constrained and searched for, these queries are classified as:

- Exact match query: The query specifies the desired values for all resource attributes sought. For example, Architecture = 'x86' and CPU-Speed = '3 Ghz' and type = 'SMP' and price = '2 Grid dollars per second' and RAM = '256 MB' and no. of processors = 10 and Secondary free space = '100 MB' and Interconnect bandwidth = '1 GB/s' and OS = 'linux' (multiple-dimension exact match query).
- Partial match query: Only selected attribute values are specified. For example, Architecture = 'sparc' and type = 'SMP' and no. of processors = 10 (multiple-dimension partial match query).
- Range queries: Range values for all or some attributes are specified. For example, Architecture = 'Macintosh' and type = 'Cluster' and '1 GHz' ≤ CPU-Speed ≤ '3 GHz' and '512MB' ≤ RAM ≤ '1 GB' (multiple-dimension range query).
- Boolean queries: All or some attribute values satisfying certain boolean conditions, such as not RAM ≤ '256 MB' and not no. of processors ≤ 5 (multiple-dimension boolean query).

P2P Taxonomy

The taxonomy for P2P-based GRIS is divided into the following (Fig. 9): P2P network organization, data organization, and *d*-dimensional query routing organization.

P2P Network Organization

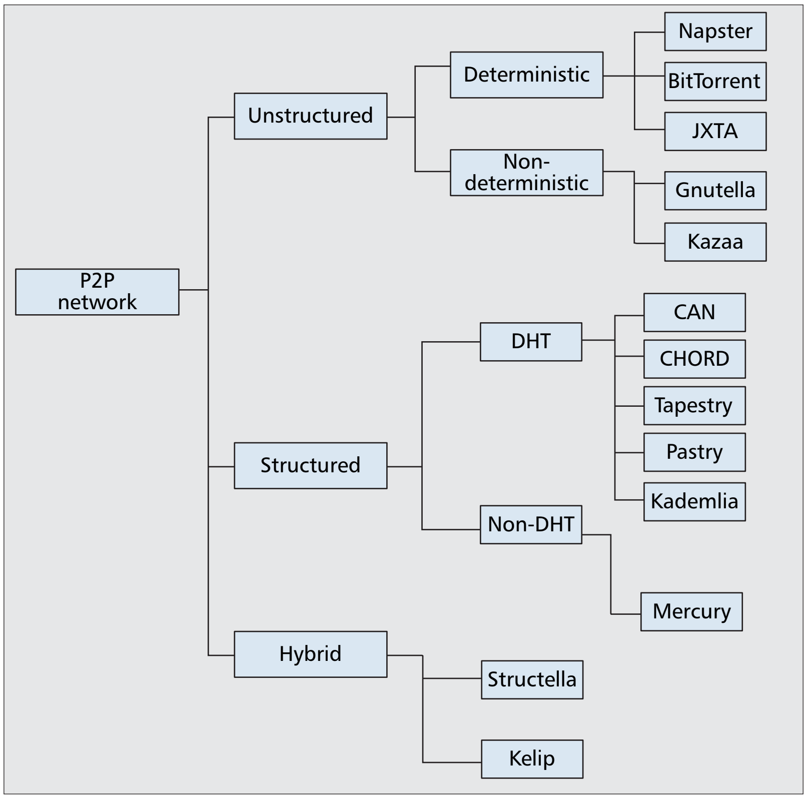
The network organization refers to how peers are logically structured from the topological perspective. Figure 10 shows the network organization taxonomy of general P2P systems. Two categories are proposed in P2P literature [1]: unstructured and structured. An unstructured system is typically described by a power law random graph model [81, 82], as peer connections are based on the popularity of content. These systems do not put any constraints on placement of data items on peers and how peers maintain their network connections. Detailed evaluation and analysis of network models [83, 84] for unstructured systems can be found in [85]. Unstructured systems including Napster, Gnutella, and Kazaa offer differing degrees of decentralization. The degree of decentralization refers to the extent peers can function independently with respect to efficient object lookup and query routing. Our taxonomy classifies unstructured systems as *deterministic* or *nondeterministic* [85].

A deterministic system means that a lookup operation will be successful within predefined bounds. Systems including Napster and BitTorrent fall into this category. In these systems the object lookup operation is centralized while download is decentralized. Under centralized organization, a specialized (index) server maintains the indexes of all objects in the system (e.g., Napster, BitTorrent). The resource queries are routed to index servers to identify the peers currently responsible for storing the desired object. The index server can obtain the indexes from peers in one of the following ways:

- Peers directly inform the server about the files they are currently holding (e.g., Napster).
- By crawling the P2P network (an approach similar to a Web search engine).

The lookup operations in these systems are deterministic and resolved with a complexity of $O(1)$. We classify JXTA as an unstructured P2P system that offers deterministic search performance. At the lowest level JXTA is a routing overlay, not unlike routers that interconnect to form a network. Hence there is no structure, but there is a routing algorithm that allows any router to router communication. In JXTA both object lookup and download operations are completely decentralized.

Other unstructured systems, including Gnutella, Freenet, FastTrack, and Kazaa, offer nondeterministic query performance. Unlike Napster or BitTorrent, both object lookup and download operation in these systems are decentralized. Each peer maintains indexes for the objects it is currently holding. In other words, indexes are completely distributed. The



■ **Figure 10.** *Peer-to-peer network organization taxonomy.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

P2P system	Overlay structure	Lookup protocol	Network parameter	Routing table size	Routing complexity	Join/leave overhead
Chord	1-dimensional, circular-ID space	Matching key and NodeID	n = number of nodes in the network	$O(\log(n))$	$O(\log(n))$	$O((\log(n))^2)$
Pastry	Plaxton style mesh structure	Matching key and prefix in NodeID	n = number of nodes in the network, b = base of the identifier	$O(\log_b(n))$	$O(b \log_b(n)+b)$	$O(\log(n))$
CAN	d-dimensional ID space	key, value pairs map to a point P in the d-dimensional space	n = number of nodes in the network, d = number of dimensions	$O(2\ d)$	$O(d\ n^{1/d})$	$O(2\ d)$
Tapestry	Plaxton style mesh structure	Matching suffix in NodeID	n = number of nodes in the network, b = base of the identifier	$O(\log_b(n))$	$O(b \log_b(n)+b)$	$O(\log(n))$

Table 3. *Summary of the complexity of structured P2P systems.*

Routing substrate	Network organization	Distributed indexing algorithm name
Chord	Structured	PHT [54], MAAN [17], Dgrid [49], Adaptive [97], DragonFly [98], QuadTree [99], Pub/Sub-2 [100], P-tree [52], Squid [21]
Pastry	Structured	XenoSearch [55], AdeepGrid [48], Pub/Sub-1 [101]
CAN	Structured	HP-protocol [22], Kd-tree [102], Meghdoot [103], Zcurve [102], Super-P2P R*-Tree [104]
Bamboo	Structured	SWORD [50]
Epidemic-DHT [95]	Hybrid	XenoSearch-II [96]
Others	Unstructured	Mercury [53], JXTA search [105], P2PR-tree [106]

Table 4. *Classification based on P2P routing substrate.*

Gnutella system employs a query flooding model for routing object queries. Every request for an object is flooded (broadcast) to directly connected peers, which in turn flood their neighboring peers. This approach is used in the GRIS model proposed by [18]. Every RLQ message has a TTL field associated with it (i.e., maximum number of flooding hops/steps allowed). Drawbacks for flood-based routing include high network communication overhead and nonscalability. This issue is addressed to an extent in FastTrack and Kazaa by introducing the notion of super-peers. This approach reduces network overhead but still uses a flooding protocol to contact super-peers.

Structured systems such as DHTs offer deterministic query search results within logarithmic bounds on network message complexity. Peers in DHTs such as Chord, CAN, Pastry, and Tapestry maintain an index for $O(\log(n))$ peers, where n is the total number of peers in the system. Inherent to the design of a DHT are the following issues [86]:

- Generation of node-ids and object-ids, called keys, using cryptographic/randomizing hash functions such as SHA-1 [87–89]. The objects and nodes are mapped on the overlay network depending on their key value. Each node is assigned responsibility for managing a small number of objects.
- Building up routing information (routing tables) at various nodes in the network. Each node maintains the network location information of a few other nodes in the network.
- An efficient lookup query resolution scheme. Whenever a node in the overlay receives a lookup request, it must be able to resolve it within acceptable bounds such as in $O(\log(n))$ time. This is achieved by routing the lookup request to the nodes in the network that are most likely to store the information about the desired object. Such probable nodes are identified by using the routing table entries.

Although at the core various DHTs (Chord, CAN, Pastry, etc.) are similar, there are still substantial differences in the actual implementation of algorithms, including the overlay network construction (network graph structure), routing table maintenance, and node join/leave handling. The performance metrics for evaluating a DHT include fault tolerance, load balancing, efficiency of lookups and inserts, and proximity awareness [90, 91]. In Table 3, we present a comparative analysis of Chord, Pastry, CAN and Tapestry based on basic performance and organization parameters. Comprehensive details about the performance of some common DHTs under churn can be found in [92].

Other classes of structured systems such as Mercury do not apply randomizing hash functions for organizing data items and nodes. The Mercury system organizes nodes into a circular overlay and places data contiguously on this ring. As Mercury does not apply hash functions, data partitioning among nodes is nonuniform. Hence, it requires an explicit load balancing scheme. In recent developments new-generation P2P

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

systems have evolved to combine both unstructured and structured P2P networks. We refer to this class of systems as hybrid. Structella [93] is one such P2P system that replaces the random graph model of an unstructured overlay (Gnutella) with a structured overlay, while still adopting the search and content placement mechanism of unstructured overlays to support complex queries. Other hybrid P2P designs include Kelips [94] and its variants. Nodes in a Kelips overlay periodically gossip to discover new members of the network, and during this process nodes may also learn about other nodes as a result of lookup communication. Another variant of Kelips [95] allows routing table entries to store information for every other node in the system. However, this approach is based on the assumption that the system experiences low churn rate [92]. Gossiping and a one-hop routing approach have been used for maintaining the routing overlay in the work [96]. In Table 4 we summarize the different P2P routing substrates utilized by the existing algorithms for organizing a GRIS.

DATA ORGANIZATION TAXONOMY

Traditionally, DHTs have been efficient for one-dimensional queries such as finding all resources that match the given attribute value. Extending DHTs to support DRQs to index all resources whose attribute values overlap a given search space, is a complex problem. DRQs are based on ranges of values for attributes rather than specific values. Compared to one-dimensional queries, resolving DRQs is far more complicated, as there is no obvious total ordering of the points in the attribute space. Furthermore, the query interval has varying size, aspect ratio, and position, such as a window query. The main challenges involved in enabling DRQs in a DHT network [102] include efficient:

- Data distribution mechanisms;
- Data indexing or query routing techniques.

In this section we discuss various data distribution mechanisms; we analyze data indexing techniques in the next section.

A data distribution mechanism partitions the d-dimensional [107, 108] attribute space over the set of peers in a DHT network. Efficiency of the distribution mechanism directly governs how the query processing load is distributed among the peers. A good distribution mechanism should possess the following characteristics [102]:

- Locality: Tuples or data points nearby in the attribute space should be mapped to the same node, hence limiting the lookup complexity.
- Load balance: The number of data points indexed by each peer should be approximately the same to ensure uniform distribution of query processing [109, 110].
- Minimal metadata: Prior information required for mapping the attribute space to the peer space should be minimal.
- Minimal management overhead: During peer join and leave operation, update policies such as the transfer of data points to a newly joined peer should cause minimal network traffic.

In the current P2P literature (see the next section) d-dimensional data distribution mechanisms based on the following structures have been proposed (Fig. 12): space filling curves, tree-based structures, and a variant of SHA-1/2 hashing. In Table 5 we summarize various data structures used in different algorithms for d-dimensional data distribution. Furthermore, in Table 6 we present a classification of the existing algorithms based on the number of routing overlays utilized for managing d-dimensional data.

The space filling curves (SFCs) data structure [111, 112] includes the Z-curve [113] and Hilbert's curve [114]. SFCs map the given d-dimensional attribute space into a one-dimensional space. The work in [22] utilizes SFCs, in particular the reverse Hilbert SFC for mapping a one-dimensional attribute space to a two-dimensional CAN P2P space. Similarly, the work in [21] uses the Hilbert SFC to map a d-dimensional index space into a one-dimensional space. The resulting one-dimensional indexes are contiguously mapped on a Chord P2P network. The approach proposed in [102] utilizes Z-curves for mapping a d-dimensional space to a one-dimensional space. SFCs exhibit the locality property by mapping the points that are close in d-dimensional space to adjacent spaces in the one-dimensional space. However, as the number of dimensions increases, locality becomes worse since SFCs suffer from the "curse of dimensionality" [115]. Furthermore, SFC-based mapping fails to uniformly distribute the load among peers if the data distribution is skewed. Hence, this leads to a nonuniform query processing load for peers in the network.

Some recent work [52, 54, 97, 99] utilizes tree-based data structures for organizing the data. The approach proposed in [99] adopts the MX-CIF quadtree [116] index for P2P networks. A distributed quadtree index assigns regions of space (a quadtree block) to the peers. If the extent of a spatial object goes beyond a quadtree block, recursive subdivision of that block can be performed. With a good base hash function one can achieve a uniform random mapping of the quadtree blocks to the peers in the network. This approach will map two quadtree blocks that are close to each other to totally different locations on the Chord space. Another recent work called DragonFly [98] uses the same base algorithm with an enhanced load balancing technique called recursive bisection [117]. Recursive bisection works by dividing a cell/block recursively into two halves until a certain load condition is met. The load condition is defined based on two load parameters known as the load limit and load threshold. Hence, this approach has better load balancing properties than the SFC-based approaches in the case of a skewed data set.

Figure 11 depicts the index space organization and mapping to the Chord overlay in the DragonFly publish/subscribe system. DragonFly builds a d-dimensional Cartesian space based on the grid resource attributes, where each attribute represents a single dimension. The logical d-dimensional index assigns regions of space to the peers. If a peer is assigned a region (index cell) in the d-dimensional space, it is responsible for handling all activities related to the subscription and publication associated with the region. Each cell is uniquely identified by its centroid, called the *control point*. Figure 11 depicts some control points and some example hashings in a two-dimensional attribute space using the Chord method.

Other approaches including [17, 49] manipulate existing SHA-1/2 hashing for mapping d-dimensional data to the peers. MAAN addresses the one-dimensional range query problem by mapping attribute values to the Chord identifier space via a uniform locality preserving hashing scheme. A similar approach is also utilized in [100]. However, this approach shows poor load balancing characteristics when the attribute values are skewed.

To conclude, the choice of data structure is directly governed by the data distribution pattern. A data structure that performs well for a particular data set may not do the same if the distribution changes. Additional techniques such as peer virtualization (as proposed in Chord) or multiple realities (as proposed in CAN) may be utilized to improve the query processing load.

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

Algorithm name	Data structure	Lookup complexity
PHT [54]	Trie	$O(\log \lvert D \rvert)$; D is the total number of bits in the binary string representation, for 1-dimensional range query
MAAN [17]	Locality preserving hashing	$O(n \times \log n + n \times s_{min})$, s_{min} is the minimum range selectivity per dimension; n total peers
Dgrid [49]	SHA-1 hashing	$O(\log_2 Y)$ for each dimension, Y is the total resource type in the system
SWORD [50]	N/A	N/A
JXTA search [105]	RDBMS	N/A
DragonFly [98]	QuadTree	$O(E[K] \times (\log_2 n + f_{max} - f_{min}))$; n is the total peers in the network; f_{max} is the maximum allowed depth of the tree, f_{min} is the fundamental minimum level, $E[K]$ is the mean number disjoint path traversed for a window query, its distribution is function of the query size
QuadTree [99]	QuadTree	$O(E[K] \times (\log 2n + f_{max} - f_{min}))$; n is the total peers in the network; f_{max} is the maximum allowed depth of the tree, f_{min} is the fundamental minimum level, $E[K]$ is the mean number disjoint path traversed for a window query, its distribution is function of the query size
Pub/Sub-2 [100]	Order preserving hashing	$1/2 \times O(\log n)$; Equality query, n is total peers, $1/2 \times O(n_s \log n$, n_s is step factor; for ranged query, in a 1-dimensional search space
P-tree [52]	Distributed B-+ tree	$O(m + \log_d n)$; n is total peers, m is number of peers in selected range, d is order of the 1-dimensional distributed B-tree
Pub/Sub-1 [101]	SHA-1 hashing	$O(n_r \log n)$; n is total peers, n_r is the number of range intervals searched in a 1-dimensional search space
XenoSearch [55]	SHA-1 hashing	N/A
XenoSearch-II [96]	Hilbert space filling curve	N/A
AdeepGrid [48]	SHA-1 hashing	N/A
HP-protocol [22]	Reverse hilbert space filling curve	N/A
Squid [21]	Hilbert space filling curve	$n_c \times O(\log n)$; n_c is the total no. of isolated index clusters in the SFC based search index space, n is the total number of peers
Mercury [53]	N/A	$O((\log n)/k)$; k Long distance links; n is total peers, in a 1-dimensional search space
Adaptive [97]	Range search tree	$O(\log R_q)$; R_q is range selectivity, in a 1-dimensional search space
Kd-tree [102]	Kd-tree, skip pointer based on skip graphs	N/A
Meghdoot [103]	SHA-1 hashing	$O(dn^{1/d})$, n is the total peers in the network, d is the dimensionality of CAN space
Z-curve [102]	Z-curves, skip pointer based on skip graphs	N/A
P2PR-tree [106]	Distributed R-tree	N/A
Super-P2P R*-Tree [104]	Distributed R*-tree	$O(E[k] \times (d/4)(n^{1/d}))$; $E[k]$ is the mean number of MBRs indexed per range query or NN query, d is the dimensionality of the indexed/CAN space, n is the number of peers in the system.

■ Table 5. *Classification based on data structure applied for enabling ranged search and lookup complexity.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

Single	Multiple
JXTA search [105], DragonFly [98], XenoSearch-II [96], SWORD [50], Squid [21], Kd-tree [102], Meghdoot [103], Zcurve [102], QuadTree [99], P2PRtree [106], AdeepGrid [48], Super-P2P R*-Tree [104], Dgrid [49]	PHT [54], MAAN [17], Adaptive [97], Pub/Sub-2 [100], P-tree [52], XenoSearch [55], Pub/Sub-1 [101], Mercury [53], HPPROTOCOL [22]

■ Table 6. *Classification based on no. of routing overlays for d-dimensional search space.*

D-Dimensional Query Routing Taxonomy

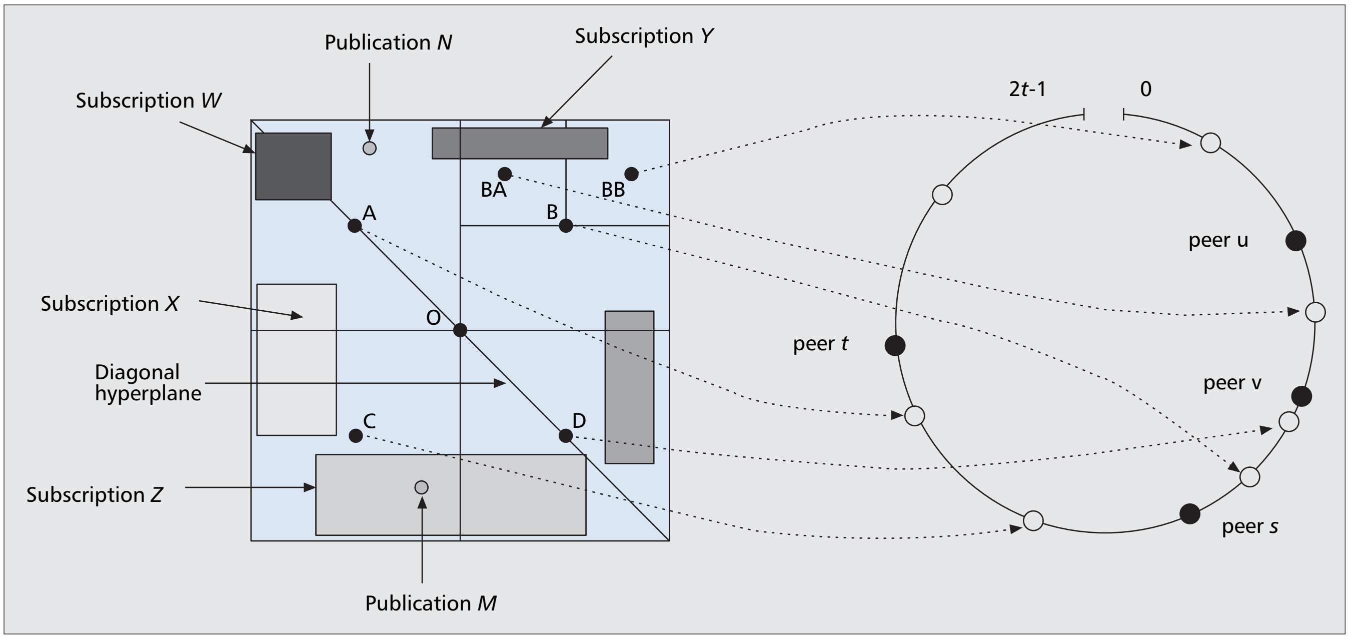
DHTs guarantee deterministic query lookup with logarithmic bounds on network message cost for one-dimensional queries. However, grid RLQs are normally DPQs or DRQs. Hence, existing routing techniques need to be augmented in order to efficiently resolve a DRQ. Various data structures discussed in the previous section effectively create a logical d-dimensional index space over a DHT network. A lookup operation involves searching for an index or set of indexes in a d-dimensional space. However, the exact query routing path in the d-dimensional logical space is directly governed by the data distribution mechanism (i.e., based on the data structure that maintains the indexes). In this context various approaches have proposed different routing/indexing heuristics. An efficient query routing algorithm should exhibit the following characteristics [102]:

- Routing load balance: Every peer in the network on average should route forward/route approximately the same number of query messages.
- Low per-node state: Each peer should maintain a small number of routing links, hence limiting new peer join and peer state update cost. In Table 5, we summarize the query lookup complexity involved with the existing algorithms.

Resolving a DRQ over a DHT network that utilizes SFCs for data distribution consists of two basic steps [21]: mapping the DRQ onto the set of relevant clusters of SFC-based index space and routing the message to all peers that fall under the computed SFC-based index space. The simulation-based study proposed in [102] has shown that SFCs (Z-curves) incur constant routing costs irrespective of the dimensionality of the attribute space. Routing using this approach is based on a skip graph, where each peer maintains $O(\log(n))$ additional routing links in the list. However, this approach has serious load balancing problems that need to be fixed using external techniques [118].

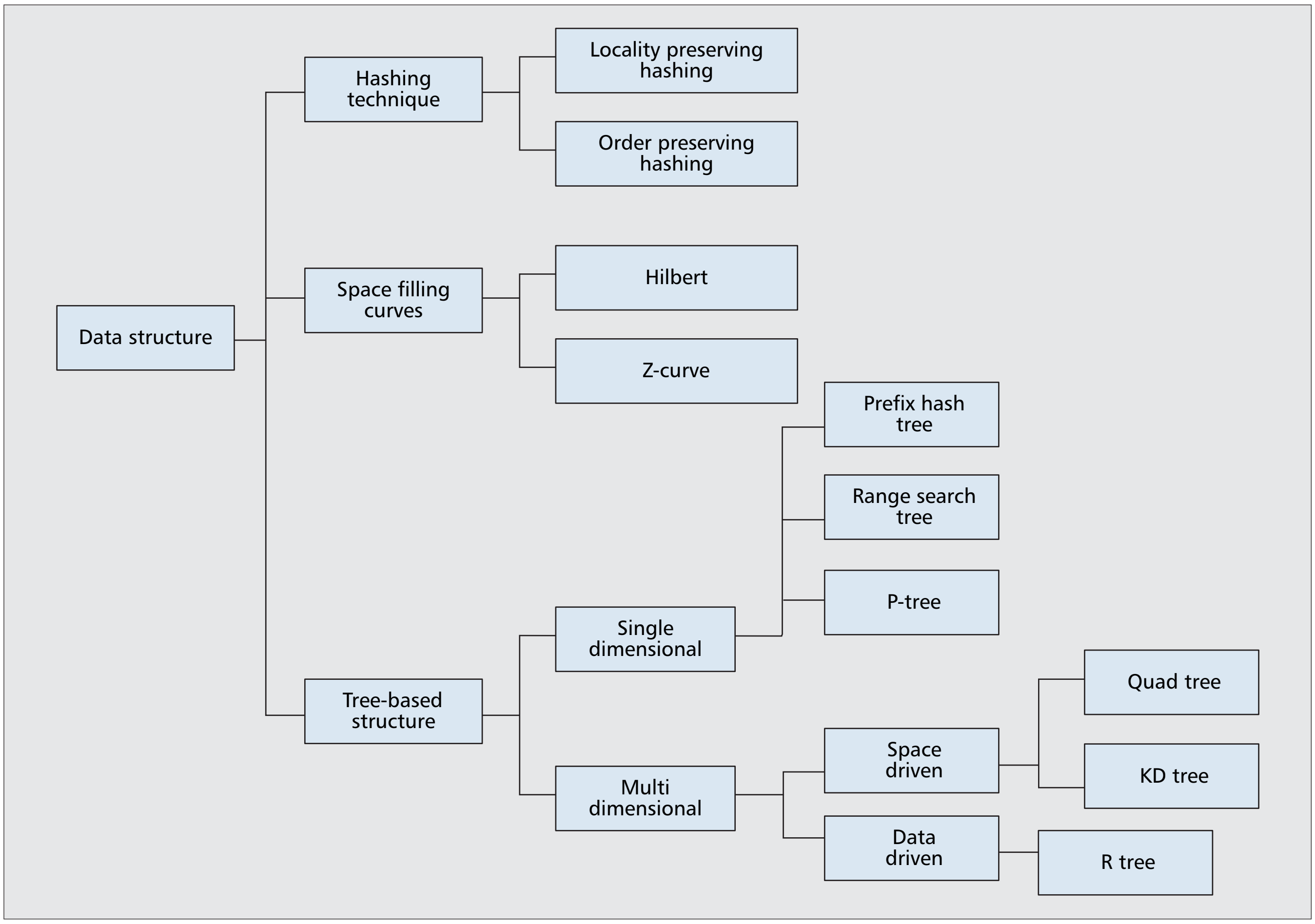
Routing DRQs in DHT networks that employ tree-based structures for data distribution requires routing to start from the root node. However, the root peer presents a single point of failure and load imbalance. To overcome this, the authors in [99] introduced the concept of a fundamental minimum level. This means that all query processing and data storage should start at the minimal level of the tree rather than at the root. Another approach [102] utilizes a P2P version of a Kd-tree [119] for mapping d-dimensional data onto a CAN P2P space. The routing utilizes the neighboring cells of the data structure. The nodes in this network that manage a dense region of space are likely to have large numbers of neighbors, hence leading to an unbalanced routing load. An example routing for this approach is shown in Fig. 13, where a query is routed from a node labeled A to its destination marked as X. Note that each node in the CAN space must know the partition boundaries of each of its neighbors for routing purposes.

Other approaches based on variants of standard hashing schemes (e.g., MAAN) apply different heuristics for resolving range queries. The single-attribute query dominated routing (SAQDR) heuristic abstracts resource attributes into two categories: dominant and nondominant attributes. The underlying system queries for the node that maintains the index information for the dominant attribute. Once such a node is found, the node searches its local index information looking at satisfying the values for other nondominant attributes in the DRQ. The request is then forwarded to the next node, which



■ **Figure 11.** *Spatial subscriptions* {W, X, Y, Z}, *cell control points, point publications* {M, N}, *and some of the hashings to the Chord, that is, the 2-dimensional coordinate values of a cell's control point is used as the key and hashed onto the Chord. Dark dots are the peers that are currently part of the network. Light dots are the control points hashed on the Chord.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.



■ **Figure 12.** *Data structure taxonomy.*

indexes the subsequent range value for the dominant attribute. This approach comprehensively reduces the number of routing steps needed to resolve a DRQ. However, this approach suffers from routing load imbalance in the case of a skewed attribute space. In Table 7 we present a classification of the existing algorithms based on query resolution heuristic and data locality preserving characteristics.

SURVEY OF P2P-BASED GRID INFORMATION INDEXING

PASTRY-BASED APPROACHES

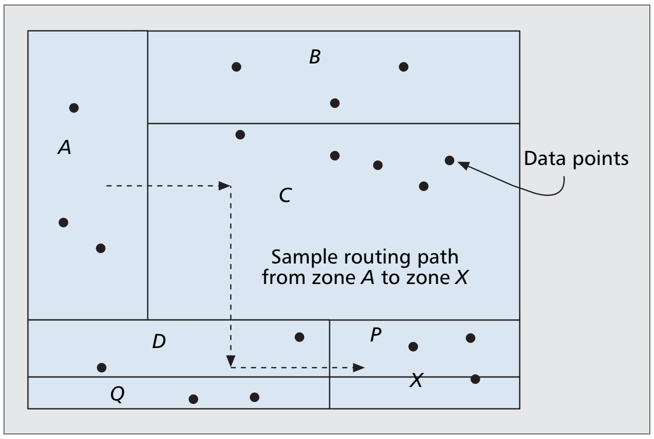
Pub/Sub-1: Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables — The content-based publish/subscribe (Pub/Sub) system [101] is built using a DHT routing substrate. They use the topic-based Scribe [45] system which is implemented using Pastry [42]. The model defines different schema for publication and subscription messages for each application domain (e.g., a stock market or an auction market). The proposed approach is capable of handling multiple domain schema simultaneously. Each schema includes several tables, each with a standard name. Each table maintains information about a set of attributes, including their type, name, and constraints on possible values. Furthermore, there is a set of *indices* defined on a table, where each index is an ordered collection of strategically selected attributes. The model requires application designers to manually specify the domain scheme.

When a request (publication or subscription) is submitted to the system, it is parsed for various index digests. An index digest is a string of characters that is formed by concatenating the attribute type, name, and value of each attribute in the index. An example index digest is [*USD* : *Price* : *100* : *Inch* : *Monitor* : *19* : *String* : *Quality* : *Used*]. Handling publication/subscription with exact attribute values is straightforward as it involves hashing the published request or subscription request. When a publication with attribute values that match a subscription is submitted to the system, it is mapped to the same hash key as the original subscription. When such Pub/Sub event matching occurs, the subscribing node is notified accordingly. The model optimizes the processing of popular subscription (many nodes subscribing for an event) by building a multicast tree of nodes with the same subscription interest. The root of the tree is the hash key's home node (the node at which publication and subscription requests are stored in the network), and its branches are formed along the routes from the subscriber nodes to the root node.

The system handles range values by building a separate index hash key for every attribute value in the specified range. This method has serious scalability issues. The proposed approach to overcome this limitation is to divide the range of values into intervals and a separate hash key is built for each such index digest representing that interval. However, this approach can only handle range values of a single attribute in an index digest (does not support multi-attribute range values in a single index digest).

XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform — XenoSearch [55] is a resource

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.



■ **Figure 13.** *Example routing in a 2-dimensional CAN space that utilizes k-d tree index to organize* d*-dimensional data. Peer X's coordinate neighbor set = {D, P, Q}; peer A's coordinate neighbor set = {B, C, D}; peer C's coordinate neighbor set = {A, B, D, P}, peer D's coordinate neighbor set = {A, C, P, X, Q}.*

discovery system built for the XenoServer [120] execution platform. The XenoServer system is an Internet-based resource sharing platform that allows users to run programs at topologically distributed nodes. The XenoSearch indexes the resource information that is advertised periodically by the XenoServers. An advertisement contains information about the identity, ownership, location, resource availability, and access prices of a XenoServer. The XenoSearch system converts these advertisements to points in a *d*-dimensional space, wherein different dimensions represent different attributes (topological location, QoS attributes, etc.). The XenoSearch system is built over the Pastry [42] overlay routing protocol.

A separate Pastry ring operates for each dimension with XenoSearch nodes registering separately in each ring. A XenoServer registers for each dimension and derives the overlay key by hashing its coordinate position in that dimension. Effectively, in different dimensions a XenoServer is indexed by different *keys*. In each dimension the resource information is logically held in the form of a tree where the leaves are the individual XenoServers and interior nodes are aggregation points (APs) that summarize the membership of ranges of nodes below them. These APs are identified by locations in the key space that can be determined algorithmically by forming keys with successively longer suffixes. The XenoSearch node closest in the key space to an AP is responsible for managing this information and dealing with messages it receives. This locality in search is provided by the proximity-aware routing characteristic of the Pastry system. The *d*-dimensional range searches are performed by making a series of search requests in each dimension and finally computing their intersection.

Recently, XenoSearch has been enhanced with a new search and data placement technique [96]. The new approach puts emphasis on both the location and resource constraints associated with a search entity. Location constraints are defined using the primitives of disjunction ((), conjunction (϶), proximity (*near*($A_1, A_2, \ldots, A_n$)), A_i denotes the *i*th resource attribute, distribution (*near*($A_1, A_2, \ldots, A_n$)), terms representing fixed locations (e.g., clients' positions in the network) and free servers to locate (i.e., the resource request terms to be matched to machines). A quadtree-based [116] data structure is used for the centralized implementation and an epidemic/gossip-based distributed data structure for the distributed resource discovery system. Gossip techniques between peer nodes separate the maintenance and distribution of summaries from the implementation of the algorithm. Nodes determine the network location of the indexed machines by using a coordinate location system [121]. These *d*-dimensional coordinates are then mapped to a one-dimensional linear index space using the Hilbert SFC.

AdeepGrid: Peer-to-Peer Discovery of Computational Resources for Grid Applications — AdeepGrid [48] presents an algorithm for grid resource indexing based on the Pastry DHT. The proposed GRIS model incorporates both static and dynamic resource attributes. A *d*-dimensional attribute space (with static and dynamic attributes) is mapped to a DHT network by hashing the attributes. The resulting key forms a Resource ID, which is also the key for the Pastry ring. The key size is 160 bits long as compared to 128 bits in the standard Pastry ring. In this case the first 128 bits are used to encode the static attributes, while the remaining 32 bits for the dynamic attributes. The static part of the Resource ID is mapped to a fixed point, while the dynamic part is represented by potentially overlapping arcs on the overlay. The beginning of each arc represents a resource's static attribute set, while the length of the arc signifies the spectrum of the dynamic states a resource can exhibit. Effectively, the circular node Id space contains only a finite number of nodes, while they store an infinite number of objects representing dynamic attributes. RUQs can be periodically initiated if the dynamic attribute value changes by a significant amount (controlled by a system-wide UCHANGE parameter). Such updates are carried out using an UPDATE message primitive. However, in some cases the new update message may map to a different node (due to a change in an attribute value) as compared to the previous INSERT or UPDATE. This can lead to defunct objects in the system. The proposed approach overcomes this by making nodes periodically flush resource entries that have not changed recently or sending REMOVE messages to prior node mappings.

Resolving RLQ involves locating the node that currently hosts the desired resource attributes (Resource ID). This is accomplished by utilizing standard Pastry routing. Three different heuristics for resolving the RLQs are proposed: single-shot searching, recursive searching, and parallel searching. Single-shot searching is applied in cases where the grid application implements local strategies for searching. In this case a query for a particular kind of resource is made, and if the search was successful, the node hosting the desired information replies with a REPLY message that contains resource information. On the other hand, recursive searching is a TTL restricted search that continuously queries the nodes that are likely to know the desired resource information. At each step the query parameters, in particular, the dynamic attribute search bits are tuned. Such a tuning can help to locate resources that may not match exactly, but are close approximations of the original requirements. Finally, the parallel search technique initiates multiple search queries in addition to a basic search for the exact match requested parameters.

CHORD-BASED APPROACHES

DGRID: A DHT-Based Grid Resource Indexing and Discovery Scheme — Work by Teo *et al.* [49] proposed a model for supporting GRIS over the Chord DHT. The unique characteristic of this approach is that the resource information is maintained in the originating domain. Every domain in DGRID designates an index server to the Chord-based GRIS network. The index server maintains state and attribute information for the local resource set. The model distributes the multi-attribute resource information over the overlay using

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

Algorithm name	Heuristic name	Preserves data locality (yes/no)
PHT [54]	Chord routing	N/A
MAAN [17]	Iterative resolution, single attribute dominated routing based on Chord	N/A
Dgrid [49]	Chord routing	N/A
SWORD [50]	Bamboo routing	No
JXTA search [105]	Broadcast	N/A
DragonFly [98]	Generic DHT routing	No
QuadTree [99]	Generic DHT routing	No
Pub/Sub-2 [100]	Chord routing	N/A
P-tree [52]	Generic DHT routing	N/A
Pub/Sub-1 [101]	Pastry routing	N/A
XenoSearch [55]	Generic DHT routing	N/A
XenoSearch-II [96]	Generic DHT routing	N/A
AdeepGrid [48]	Single shot, recursive and parallel searching based on Pastry	No
HP-protocol [22]	Brute force, controlled flooding, directed controlled flooding based on CAN	N/A
Squid [21]	Generic DHT routing	Yes
Mercury [53]	Range-selectivity based routing	N/A
Adaptive [97]	Generic DHT routing	N/A
Kd-tree [102]	Skip pointer based routing	Yes
Meghdoot [103]	CAN based routing	Yes
Z-curve [102]	Skip pointer based routing	Yes
P2PR-tree [106]	Block/group/subgroup pointer based routing	Yes
Super-P2P R*-Tree [104]	CAN based routing	Yes

Table 7. *Classification based on query resolution heuristic, data distribution efficiency, and data locality preserving characteristic.*

the following schemes: a computational grid domain is denoted by $G = \{d\}$, where d is an administrative domain. Every domain $d = \{S, R, T\}$, consists of S; an index server such as MDS [77], R; a set of compute resources, and $T = \{a\}$; different resource type set, where $a = \{attr\_type, attr\_value\}$ (e.g., $\{CPU - Speed$, 1.7GHz$\}$). An index server S maintains indices to all the resource types in its home domain, $S = \{r_1, r_2, \ldots, r_n\}$. An index r is defined as $r = \{t, d\}$, which denotes that r is a pointer to a resource type t. There is a one-to-one relationship between S and T.

The DGRID avoids node identifier collisions by splitting it into two parts: a prefix that denotes a data identifier r and a suffix that denotes an index-server identifier S. Given a node n representing $r = (t, d)$, the m-bit identifier of n is the combination of the i-bit identifier of t, where $i \le m$, and the $m - i$ bit identifier of S. So effectively, $id_m(n) = id_i(t) \oplus id_{m-i}(S)$. Hence, DGRID guarantees that all $id_m(n)$ are unique, given that the identifiers of two nodes differ in either prefixes or suffixes. The system initialization process requires the index server S to perform the virtualization of its indices onto T virtual servers. Each virtual server joins the DGRID system to become an overlay Chord node. This process is referred to as a *join*.

The search or lookup operation in the DGRID is based on Chord lookup primitives. Given a key, p, it is mapped to a particular virtual index server on the overlay network using the query $get(p)$. The DGRID indexing approach also supports domain-specific resource type search. To facilitate such a lookup operation, index S for domain d is identified by $id'_{m-i}(S) = id_j(d) \oplus id_{m-i-j}(S), j < (m - i)$. In this case a query for resource n of type t is routed to a node n that maps to S, where $prefix_j(id'_{m-i}(S)) = id_j(d), d \in D$. In general, a query q to look up a resource type t is translated to the query q', $id_m(q') = id_i(t) \oplus 0$. This is done as $id(t)$ is i-bit length, whereas the

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

identifier space is m-bit long. Overall, the lookup cost is bounded by the underlying Chord protocol: $O(\log N)$. In general, the lookup cost for a particular resource type t is $O(\log Y)$; Y is the total number of resource types available in the network.

Adaptive: An Adaptive Protocol for Efficient Support of Range Queries in DHT-Based Systems — The work in [97] presents an algorithm to support range queries based on a distributed logical range search tree (RST). Inherently, the RST is a complete and balanced binary tree with each level corresponding to a different data partitioning granularity. The system abstracts the data being registered and searched in the network as a set of attribute-value pairs (AV-pairs): $\{a_1 = v_1, a_2 = v_2, \ldots, a_n = v_n\}$. It utilizes Chord for distributed routing and network management issues. A typical range query with length R_q is resolved by decomposing it into $O(\log(R_q))$ subqueries. These subqueries are then sent to the nodes that index the corresponding data. The system supports updates and queries for both static and dynamic resource attributes.

The content represented by an AV-pair is registered with the node whose ID is numerically closest to the hash of the AV-pair. To overcome the skewed distribution, the system organizes nodes in a logical load balancing matrix (LBM). Each column in the LBM represents a partition (i.e., a subset of content names that contain a particular AV-pair), while nodes in different rows within a column are replicas of each other. Initially, an LBM has only one node, but whenever the registration load on a particular node in the system exceeds a threshold (T_{reg}), the matrix size is increased by 1. All future registration requests are shared by the new nodes in the LBM. Note that the number of partitions P is proportional to the registration load,

$$P = \left\lceil \frac{L^R}{C_R} \right\rceil,$$

where L^R is the data item's registration load, and C_R is the capacity of each node.

An attribute a can have numerical values denoted by domain D_a. D_a is bounded and can be discrete or continuous. D_a is split up into subranges and assigned to different levels of the RST. An RST with n nodes has $O(\lceil \log n + 1 \rceil)$ levels. Levels are labeled consecutively with the leaf level being level 0. Each node in the RST holds indexing information for different subranges. Typically, the range of the ith node from the left represents the range $[v_i, v_{i+2l-1}]$. The union of all the ranges at each level covers the full D_a. In a static RST, the attribute value v is registered at each node in the tree that lies on the path $path(v)$ to the leaf node that indexes the exact value. The new value information is updated into the LBM if a node on the path maintains it.

In a static setting, a query $Q : [s, e]$ for values of an attribute a is decomposed into k subqueries, corresponding to k nodes in the RST, $N_1, \ldots, N_k$. The efficiency of the query resolution algorithm depends on the *relevance* factor, which is given by

$$r = \frac{R_q}{\sum_{i=1}^{k} R_i},$$

where R_i is node N_i's range length, and R_q is the query length. The relevance factor r denotes how efficiently the query range matches the RST nodes that are being queried. The query Q is resolved by querying the node that has the largest range within $[s, e]$ (also referred to as the node that has the minimum cover [MC] for the query range). Furthermore, this process is recursively repeated for the segments of the range that are not yet decomposed. When the MC is determined, the query is triggered on all the overlay nodes that correspond to each MC node. For dynamic setting, the authors proposed additional optimization and organization techniques, more details on these aspects of the system can be found in the referenced article.

Pub/Sub-2: Content-based Publish-Subscribe over Structured P2P Networks — The work in [100] presents a content-based publish-subscribe indexing system based on the Chord DHT. The system is capable of indexing d-dimensional index space by having a separate overlay for each dimension. Every ith dimension or an attribute a_i has a distinct data type, name, and value $v(a_i)$. An attribute type belongs to a predefined set of primitive data types commonly defined in most programming languages. An attribute name is normally a string, whereas the value can be a string or numeric in any range defined by the minimum and maximum ($v_{min}(a_i)$, $v_{max}(a_i)$) along with the attribute's precision $v_{pr}(a_i)$. The model supports a generalized subscription schema that includes different data sets and constraints on their values such as $=$, $\neq$, $>$, $<$. With every subscription, the model associates a unique subscription identifier (subID). The subID is the concatenation of three parts — c_1, c_2 and c_3. c_1 is the ID of the node receiving the subscription; the number of bits in the subID is equal to the m-bits in the Chord identifier space. c_2 is the ID of the subscription itself, and c_3 is the number of attributes on which the constraints are declared.

An attribute a_i of a subscription with identifier subID is placed on a node $successor(h(v(a_i)))$ in the Chord ring. A subscription can declare a range of values for the attribute, a_i, such as $v_{low}(a_i)$ and $v_{high}(a_i)$. In this case the model follows n_s steps, where

$$n_s = \frac{v_{high}(a_i) - v_{low}(a_i)}{v_{pr}(a_i)};$$

at each step a Chord node is chosen by the $successor(h(v_{low}(a_i) + v_{pr}(a_i)))$ function. In the subsequent steps the previous attribute value is incremented by the precision value $v_{pr}(a_i)$ and mapped to the corresponding Chord node. Updating the range values is done by following the same procedure for all Chord nodes that store the subID for the given range of values. The overall message routing complexity depends on the type of constraints defined over the attributes for a given subID. In case of equality constraints, the average number of routing hops is $O(1/2 \log(n))$. When the constraint is a range, the complexity involved is $O(n_s \times 1/2 \log(n))$, where n is the step factor.

An information publish event in the system is denoted by $N_{a-event}$ that includes various attributes with search values. An event-publish to event-notify matching algorithm processes each attribute associated with $N_{a-event}$ separately. It locates various nodes that store the subIDs for an attribute a_i, by applying the function $successor(h(v(a_i)))$. The matching algorithm then stores the list of unique subIDs that are found at a node n in the list L_{ai} designated for a_i. The N_{k-sub} list stores the subIDs that match the event $N_{a-event}$. A $subID_k$ matches an event if and only if it appears in exactly N_{k-sub} derived from the different Chord ring. The overall message routing complexity involved in locating the list of subIDs matching an event $N_{a-event}$ is $O(1/2 \log(n))$. The authors also propose a routing optimization technique to reduce the lookup search complexity.

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

QuadTree: Using a Distributed Quadtree Index in Peer-to-Peer Networks — The work in [99] proposes a distributed quadtree index that adopts an MX-CIF quadtree [116] for accessing spatial data or objects in P2P networks. A spatial object is an object with extents in a d-dimensional setting. A query that seeks all the objects contained in or overlapping a particular spatial region is called a spatial query. Such queries are resolved by recursively subdividing the underlying d-dimensional space and then solving a possibly simpler intersection problem. This recursive subdivision process utilizes the basic quadtree representation. In general, the term quadtree is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of common decomposition of space.

The work builds on the region quadtree data structure. In this case, by applying the fundamental quadtree decomposition property, the underlying two-dimensional square space is recursively decomposed into four congruent blocks until each block is contained in one of the objects in its entirety or is not contained in any of the objects. The distributed quadtree index assigns regions of d-dimensional space to the peers in a P2P system. Every quadtree block is uniquely identified by its centroid, termed the control point. Using the control point, a quadtree block is hashed to a peer in the network. The Chord method is used for hashing the blocks to the peers in the network. If a peer is assigned a quadtree block, it is responsible for processing all query computations that intersect the block. Multiple control points (i.e., quadtree blocks) can be hashed to the same peer in the network. To avoid a single point of failure at the root level of the quadtree, the authors incorporate a technique called *fundamental minimum level*, f_{min}. This technique means that objects are only allowed to be stored at levels $l \geq f_{min}$; therefore, all the query processing starts at levels $l \geq f_{min}$. The scheme also proposes the concept of a *fundamental maximum level*, f_{max}, which limits the maximum depth of the quadtree at which objects are inserted.

A peer initiates a new object insertion or query operation by calling the methods InsertObject() or ReceiveClients-Query(). These methods in turn call a subdivide() method that computes the intersecting control point associated with the new object or lookup query. Once the control points are computed, the peer broadcasts the insertion or query operation to the peer(s) that own(s) the respective control points. The contacted peers evoke DoInsert() and Do-Query() methods to determine the location for the inserted object or to locate the peers that can answer the query. The operation may propagate down to the f_{max} level or until all relevant peers are located. The authors also propose some optimizations such as each node maintaining a cache of addresses for its immediate children in the hierarchy. This reduces the subsequent lookup complexity to $O(1)$ beyond the root peer at the f_{min} level, as it is no longer required to traverse the Chord ring for each child. However, this is only true when the operation is a regular tree traversal. Note that on average, $O(\log n)$ messages are required to locate a root peer for a query.

DragonFly: A Publish-Subscribe Scheme with Load Adaptability — The work in [111] proposes a content-based publish-subscribe system with load adaptability. They apply a spatial hashing technique for assigning data to the peers in the network. The system supports multi-attribute point and range queries. The query routing and object location (subscription and publication) mechanism can be built using the services of any DHT. Each distinct attribute is assigned a dimension in a d-dimensional Cartesian space. Hence, a domain with d attributes $\{A_1, A_2, \ldots, A_d\}$ will be represented by a d-dimensional Cartesian space. Every attribute in the system has lower and upper bounds on its values. The bounds act as constraints for subscriptions and event indexing. The d-dimensional Cartesian space is arranged as a tree structure with the domain space mapped to the root node of the tree. In particular, the tree structure is based on a quad tree [116]. To negate a single point of failure at the root node, the system adopts a technique called the *fundamental minimum level*. More details about this technique can be found in [99]. This technique recursively divides the logical space into four quadrants. With each recursion step on an existing quadrant, four new quadrants are generated. Hence, multiple recursion steps basically create a multilevel quadtree data structure. The quadtree-based organization of DragonFly introduces parent-child relationships between tree cells. A cell at level d is always a child of a particular cell at level $d - 1$. However, this relationship exists between consecutive levels only. In other words, every cell has a direct relationship with its child cells and no relationship with its grandchild cells. Another important feature of DragonFly is the diagonal hyperplane. This hyperplane is used to handle publish and subscribe region pruning and selection in d-dimensional space. In $2d$ space, the diagonal hyperplane is a line spanning from the northwest to the southeast vertices of the rectangular space. In the d-dimensional context, this hyperplane is represented by the equation

$$\frac{x_1}{x_{max_1} - x_{min_1}} + \frac{x_2}{x_{max_2} - x_{min_2}} + \cdots + \frac{x_d}{x_{max_d} - x_{min_d}} = K,$$

where x_{maxd} and x_{mind} are the upper and lower boundary values for the dth attribute in the domain space.

The d-dimensional domain space acts as the basis for object routing in DragonFly. Every subscription is mapped to a particular cell or set of cells in the domain space. In this case the cell acts as the subscription container. A point subscription takes the form $\{A_1 = 10, A_2 = 5\}$, while a range subscription is represented by $\{A_1 \leq 10, A_2 \leq 5\}$. The root cells at the fundamental minimum level are the entry points for a subscription's object routing. These root cells are managed by the peers in the network. Every subscription is mapped to a particular region in the d-dimensional space. The peer responsible for the region (root cell) is located by hashing the coordinate values. If the root cell has undergone the division process due to overload, the child cells (peers at a lower level in the hierarchy) are searched using the DHT routing method. Once a child cell is located, the root cell routes the subscription message to it. This process is repeated until all relevant child cells are notified for this subscription. However, if the root cell has not undergone any division process, it is made responsible for this subscription.

Mapping publication events to the peers in the network is similar to the subscription mapping process. There are two kinds of publishing events: point and range. Mapping point events is straightforward, as the relevant root cell (peer) is located by hashing the coordinated values. Resolving cells corresponding to range events can be complex. In this case the routing system sends out the published event to all the root cells that intersect with the event region. When the message reaches the root cells, a method similar to the one adopted in case of subscription events is applied to locate the child cells.

MAAN: A Multi-Attribute Addressable Network for Grid Information Services — Cai *et al.* [17] present a multi-attribute addressable network (MAAN) approach for enabling a GRIS. They extend the Chord [41] protocol to support DRQs. MAAN addresses the d-dimensional range query problem by mapping the attribute values to the Chord identi-

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

fier space via a uniform locality preserving hashing. Note that for every attribute dimension, a separate Chord overlay is maintained. For attributes with numerical values, MAAN applies locality preserving hashing functions to assign an identifier in the m-bit identifier space. A basic range query includes the numeric attribute values v between l and u for attribute a, such that $l \leq v < u$, where l and u are the lower and upper bounds, respectively. In this case node n formulates a lookup request and uses the underlying Chord routing algorithm to route it to node n_l such that $n_l = successor(H(l))$. The lookup is done using the *SEARCH_REQUEST*(k, R, X) primitive, k = successor($H(l)$) is the key to look up, R is the desired attribute value range $[l, u]$, and X is the list of resources that has the required attributes in the desired range. A node n_l, after receiving the search request message, indexes its local resource list entries and augments all the matching resources to X. In case n_l is the *successor*($H(u)$), it sends a reply message to the node n. Otherwise, the lookup request message is forwarded to its immediate successor until the request reaches node n_u, the successor of $H(u)$. The total routing complexity involved in this case is $O(\log N + K)$, where $O(\log N)$ is the underlying Chord routing complexity and K is the number of nodes between n_l and n_u.

MAAN also supports multi-attribute query resolution by extending the above single-attribute range query routing algorithm. The system maintains a separate overlay/mapping function for every attribute a_i. In this case each resource has M attributes $a_1, a_2, \ldots, a_m$ and corresponding attribute value pairs $< a_i, v_i >$, such that $1 \leq i \leq M$. Each resource registers its information (attribute value pairs) at a node n_i = *successor*($H(v_i)$) for each attribute value v_i. Thus, each node in the overlay network maintains the resource information in the form of $< attribute\text{–}value, resource\text{–}info >$ for different attributes. The resource lookup query in this case involves a multi-attribute range query, which is a combination of subqueries on each attribute dimension, that is, $v_{il} \leq a_i \leq v_{iu}$, where $1 \leq i \leq M$, and v_{il} and v_{iu} are the lower and upper bounds of the lookup query. MAAN supports two routing algorithms to resolve multiple-attribute queries:
- Iterative query resolution (IQR)
- Single attribute dominated query resolution (SADQR)

The overall routing complexity with IQR is $O(\sum_{i=1}^{M}(\log N + N \times s_i))$, while using the SAQDR technique, the lookup can be resolved in $O(\log N + N \times S_{min})$, where S_{min} is the minimum selectivity for all attributes.

Squid: Flexible Information Discovery in Decentralized Distributed Systems — Schmidt *et al.* [21] proposed a GRIS model that utilizes SFCs for mapping d-dimensional attribute space to a one-dimensional search space. The proposed GRIS model consists of the following main components:
- A locality preserving mapping that maps data elements to indices;
- An overlay network topology;
- A mapping from indices to nodes in the overlay network;
- A load balancing mechanism;
- A query engine for routing and efficiently resolving attribute queries using successive refinements and pruning.

All data elements are described using a sequence of attributes such as memory, CPU speed, and network bandwidth. The attributes form the coordinates of a d-dimensional space, while the data elements are the points. This mapping is accomplished using a locality-preserving mapping called space filling curves (SFCs) [111, 112]. SFCs are used to generate a 1d index space from the d-dimensional attribute space, where d is the number of different attribute types. Any range query or query composed of attributes, partial attributes, or wild cards can be mapped to regions of the attribute space and subsequently to the corresponding clusters in the SFC.

The Chord protocol is utilized to form the overlay network of peers. Each data element is mapped, based on its SFC-based index or key, to the first node whose identifier is equal to or follows the key in the identifier space. The lookup operation involving partial queries and range queries typically requires interrogating more than one node, since the desired information is distributed across multiple nodes. The lookup queries can consist of combinations of attributes, partial attributes, or wildcards. The result of the query is a complete set of data elements that matches the user's query. Valid queries include (computer, network), (computer, net*) and (comp*,*). The range query consists of at least one dimension that needs to be looked up for range values. The query resolution process consists of two steps:
- Translating the attribute query to relevant clusters of the SFC-based index space;
- Querying the appropriate nodes in the overlay network for data elements.

The system also supports two load balancing algorithms in the overlay network. The first algorithm proposes exchange of information between neighboring nodes about their loads. In this case the most loaded nodes give part of their load to their neighbors. The cost involved in this operation at each node is $O(\log_2^2 N)$ messages. The second approach uses a virtual node concept. In this algorithm each physical node houses multiple virtual nodes. The load at a physical node is the sum of the load of its virtual nodes. In case the load on a virtual node exceeds predefined threshold value, the virtual node is split into more virtual nodes. If the physical node is overloaded, one or more of its virtual nodes can migrate to less loaded neighbors or fingers. Note that creation of a virtual node is inherent to the Chord routing substrate.

P-Tree: Querying Peer-to-Peer Networks Using P-Trees — Crainniceanu *et al.* [52] propose a distributed fault-tolerant P2P index structure called P-tree. The main idea behind the proposed scheme is to maintain parts of semi-independent B^+–trees at each peer. The Chord protocol is utilized as a P2P routing substrate. Every peer in the P2P network believes that the search key values are organized in a ring, with the highest value wrapping around to the lowest value. Whenever a peer constructs its search tree, the peer pretends that its search key value is the smallest value in the ring. Each peer stores and maintains only the *leftmost root-to-leaf path* of its corresponding B^+–tree. The remaining part of the subtree information is stored at a subset of other peers in the overlay network. Furthermore, each peer only stores tree nodes on the root-to-leaf path, and each node has at most $2d$ entries. In this case the total storage requirement per peer is $O(d \log_d N)$. The proposed approach guarantees $O(\log_d N)$ search performance for equality queries in a consistent state. Here d is the order of the subtree, and N is the total number of peers in the network. Overall, in a stable system when no insert or delete operation is being carried out, the system provides $O(m + \log_d N)$ search cost for range queries, where m is the number of peers in the selected range in 1d space.

The data structure for a P-tree node p is a double indexed array $p.node[i][j]$, where $0 \leq i \leq p.maxLevel$ and $0 \leq j \leq p.node[i].numEnteries$, *maxLevel* is the maximum allowed height of the P-tree, and *NumEnteries* is the number of entry allowed per node. Each entry of this $2d$ array is a pair (*value*, *peer*), which points to the *peer* that holds the data item with the search key *value*. In order that the proposed scheme works properly, the P-tree should satisfy the four predefined

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

properties. These properties include the constraints on the number of entries allowed per node, left-most root-to leaf path, coverage and separation of sub-trees. The coverage property ensures that there are no gaps between the adjacent sub-trees. While the separation property ensures that the overlap between adjacent subtrees at a level i have at least d nonoverlapping entries at level $i - 1$. This ensures that the search cost is $O(\log_d N)$.

CAN Based Approaches

One Torus to Rule Them All (Kd-Tree and Z-Curve based indexing) — The work in [102] proposes two approaches for enabling DRQs over the CAN DHT. The d-dimensional data is indexed using the well known spatial data structures: z-curves and Kd-tree. First scheme is referred to as SCRAP: Space Filling Curves with Range Partitioning. SCRAP involves two fundamental steps: the d-dimensional data is first mapped to a 1-dimensional using the z-curves, and then 1-dimensional data is contiguously range partitioned across peers in the DHT space. Each peer is responsible for maintaining data in the contiguous range of values. Resolving DRQs in SCRAP network involves two basic steps: mapping DRQ into SRQ using the SFCs, and routing the 1-dimensional range queries to the peers that indexes the desired look-up value. For routing query in 1-dimensional space the work proposes a scheme based on skip graph [123]. A skip graph is a circular linked list of peers, which are organized in accordance with their partition boundaries. Additionally, peers can also maintain skip pointers for faster routing. Every peer maintains skip pointers to $O(\log(n))$ other peers at an exponentially increasing distances from itself to the list. A SRQ query is resolved by the peer that indexes minimum value for the desired range. The message routing is done using the skip graph peer lists.

Other approach referred to as d-dimensional Rectangulation with Kd-trees (MURK). In this scheme, d-dimensional space (for instance a $2d$ space) is represented as "rectangles" i.e., (hypercuboids in high dimensions), with each node maintaining one rectangle. In this case, these rectangles are used to construct a distributed Kd-tree. The leaf node in the tree are stored by the peers in the network. Routing in the network is based on the following schemes:

- CAN DHT is used as basis for routing the DRQs.
- Random pointers-each peer has to maintain skip pointers to random peers in the network. This scheme provides similar query and routing efficiency as multiple realities in CAN; and space-filling skip graph-each peer maintain skip pointers to $O(\log(n))$ other peers at exponentially increasing distances from itself in the network. Simulation results indicate that random and skip-graph based routing outperforms the standard CAN based routing for DRQs.

Meghdoot: Content-Based Publish/Subscribe over P2P Networks — The work in [103] proposes a content-based Pub/Sub system based on CAN routing substrate. Basic models and definitions are based on the scheme proposed in the work [123]. The model defines a d-dimensional attribute space given by the set $S = A_1, A_2, \ldots, A_d$. Further, each attribute value A_i is denoted using the tuple Name: Type, Min, Max. Different Type includes an integer, floating point and string character. While Min and Max denotes the range over which values lie. All peers in the system use the same schema S.

Typically, a subscription is a conjunction of predicates over one or more attributes. Each predicate specifies a constant value or range using the operators (such as =, ≥, ≤, ≥ and ≤) for an attribute. An example subscription is given by $S = (A_1 \geq v_1) \wedge (v_2 \leq A_3 \leq v_3)$. A system consisting of d attributes is always mapped to a Cartesian space of $2d$ dimensions. An attribute A_i with domain value $[L_i, H_i]$ corresponds to dimensions $2i - 1$ and $2i$ in a 2-dimensional cartesian space. The $2d$ dimensional logical space is partitioned among the peers in the system. A subscription S for d attributes is mapped to the point $< l_1, h_1, l_2, h_2, \ldots, l_d, h_d >$ in the $2d$ dimensional space which is referred to as the subscription point. Pub/Sub applications submit their subscription to a randomly chosen peer P_0. An origin peer P_0 routes the subscription request to the target peer P_t using the basic CAN routing scheme. The peer P_t owns a point in the d-dimensional space to which a subscription S maps. The overall complexity involved in routing a subscription is $O(d\, n^{1/d})$, where n is the number of peers in the system and d is the dimensionality of the Cartesian space.

Similarly every publish event is mapped to a particular point in the d-dimensional space, also referred to as the event point/event zone. The event is then routed to the P_t from the origin peer using the standard CAN routing. All the peers that own the region affected by an event are notified accordingly. Following this, all the peers in the affected region matches the new event against the previously stored subscriptions. Finally, the event is delivered to applications that have subscribed for the event.

HP-protocol: Scalable, Efficient Range Queries for Grid Information Services — Andrejak *et al.* [22] extend the CAN routing substrate to support 1-dimensional range queries. They apply the SFC in particular the Hilbert Curves for mapping a 1-dimensional attribute space (such as no. of processors) to a d-dimensional CAN space. For each resource attribute/dimension a separate CAN space is required. To locate a resource based on multiple attributes, the proposed system iteratively queries for each attribute in different CAN space. Finally, the result for different attributes are concatenated similar to "join" operation in the database.

The resource information is organized in pairs (attribute-value, resource-ID), are referred to as objects. Thus, in this case there is one object per resource attribute. Hence, if a resource has m attributes then there would be m different *object* type. The range of an attribute lies in the interval [0.0, 1.0]. A subset of the servers are designated as information servers in the underlying CAN-based P2P network (for e.g., one information server per computational resource or storage resource domain). Each of them is responsible for a certain sub-interval of [0.0, 1.0] of the attribute values. Such servers are called interval keeper (IK). Each computational resource server or storage server in the Grid registers its current attribute value to an IK. Each IK owns a zone in the logical d-dimensional Cartesian space (or a d-torus).

The CAN space is partitioned into zones, with a node (in this case an information server) serving as a zone owner. Similarly, objects (in this case (attribute, value) pair) is mapped to logical points in the space. A node R is responsible for all the objects that are mapped to its zone. It is assumed that the dimension d and the Hilbert Curve's approximation level is 1 are fixed, and known throughout the network. Given a (attribute, value) pair, a hypercube is determined by the Hilbert Function, the function returns the corresponding interval that contains the value. Following this, the message containing this object is routed to an IK whose zone encompasses this hypercube.

Given a range query r with lower and upper bounds $\in [l, u]$, a query message is routed to an information server which is responsible for the point

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

$$\frac{l+u}{2}.$$

Once such a server is located, then the request is recursively flooded to all its neighbors until all the IKs are located. Three different kinds of message flooding scheme are presented including the brute force, controlled flooding and directed control flooding. Each of these schemes has different search strategy and hence have different message routing complexities. The system handles server failures/dynamicity by defining an information update interval. If the update for one of the objects is not received in the next reporting round, the corresponding object is erased/removed from the network. In case, the object value changes (attribute value) to the extent that it is mapped to a new IK then previous object is erased in the next reporting round.

Super-P2P R*-Tree: Supporting Multi-dimensional Queries in P2P Systems — The authors in the work [104] extend the d-dimensional index R*-tree [124], for supporting range and k-Nearest Neighbor (*kNN*) queries in a super-peer [125] based P2P system. The resulting distributed R*-tree is referred to as an NRtree. Routing in the distributed d-dimensional space is accomplished through the CAN protocol. The d-dimensional distributed space is partitioned among the super-peer networks based on the Minimum Bounding Rectangle (MBR) of objects/points. Each partition (super-peer network) refers to an index-cluster (i.e., a MBR), and can be controlled by one or more super-peer. Effectively, an index-cluster includes a set of passive peers and super-peers. Evey index cluster maps to a zone in the CAN based P2P space. The functionality of a super-peer is similar to a router, it keep tracks of other index-clusters, performs inter-cluster routing, indexes data in other super-peer partition and maintains cluster-specific NR-tree. Every passive peer joins the network by contacting any available super-peer. The contacted super-peer routes the join request to other super-peer, which is responsible for the zone indexed by the passive peer. Every passive peer maintains a part of the cluster-specific NR-tree.

The bulk of query processing load is coordinated by super-peers. Super-peers can forward query to its passivepeers, in case the indexed data is managed by them. Every look-up request is forwarded to the local super-peer, which in turn forwards to other super-peers, if the requested indices are not available in the local zone. Peers initiating range query usually send the look-up rectangle, while in case of a kNN query, query point and the desired number of nearest neighbors (k). In case of a range query, the contacted super-peer routes the query to the index-cluster where the centroid of the query maps to. The owner of this index-cluster is referred to as *primary* super-peer. The primary super-peer searches its NR-tree and finds passive peers with index intersecting the query region. The passive peers directly reply to the query initiating peer when a match occurs. Every look-up query has a TTL factor, which controls the life time for a query in the network. kNN query resolution process follows a recursive path, at every successful match the *min_dist* (distance from the query point) is updated with a new value. The kNN resolution process starts at root level of NR-tree, sorting entries by their *min_dist* to query point, and then recursively traverses subtree of entries with minimum *min_dist*.

Miscellaneous

SWORD: Distributed Resource Discovery on PlanetLab — SWORD [50] is a decentralized resource discovery service that supports multi-attribute queries. This system is currently deployed and tested over PlanetLab resource sharing infrastructure. It supports different kind of query composition including per-node characteristics such as load, physical memory, disk space and inter-node network connectivity attributes such as network latency. The model abstracts resource as a networks of interconnected resource groups with intra-group, inter-group, and per-node network communication attributes. In particular, SWORD system is a server daemon that runs on various nodes. The main modules of the daemon includes the distributed query processor (DQP) and the query optimizer (QO). SWORD system groups the nodes into two sets. One set of nodes called server nodes form the part of the structured P2P overlay network [53, 126] and are responsible for managing the distributed resource information. While other set of nodes are computation nodes that report their resource attribute values to these server nodes.

For each resource attribute A_i, a corresponding DHT key k_i is computed using the standard SHA-1 scheme. A key k_i is computed based on the corresponding value of A_i at the time attribute value is sent. Each attribute is hashed to a 160-bit DHT key. The mapping function convert attribute values from their native data-type (String) and range (numeric) to a range of DHT keys. On receiving the attribute value tuple, the server node stores the tuple in the local table. In case, these values are not updated within timeout interval then are deleted (assuming node has probably left the network or owner of the key has changed due to change in attribute values). SWORD resolves multi-attribute range query similar to [53].

Users in general specify resource measurements values including the node characteristics and inter/intra-node network latency. A query also includes the node characteristics such as penalty levels for selecting nodes that are within the required range but outside the preferred range. These queries are normally written in Extended Markup Language (XML). A user submits query to a local DQP which in turn issues a distributed range query. Once the result is computed, then it is passed on to the QO (the nodes in result that are referred as "candidate nodes"). The QO selects those candidate nodes which has least penalty and passes the refined list to the user.

Mercury: Supporting Scalable Multi-Attribute Range Queries — Mercury [53] is a distributed resource discovery system that supports multi-attribute based information search. Mercury handles multi-attribute lookups by creating a separate routing hub for every resource dimension. Each routing hub represents a logical collection of nodes in the system and is responsible for maintaining range values for a particular dimension. Thus, hubs are basically orthogonal dimensions in the d-dimensional attribute space. Further, each hub is part of a circular overlay network. Mercury system abstracts the set of attributes associated with an application by $\mathcal{A}$. $\mathcal{A}_Q$ and denotes the set of attributes in a query message using Q. Attribute set for data-record $\mathcal{D}$ is denoted by $\mathcal{A}_\mathcal{D}$. The function π_a returns the value (range) for a particular attribute a in a query. An attribute hub for an attribute a is denoted by H_a. Each node in a H_a is responsible for a contiguous range r_a of values. Ranges are assigned to different overlay nodes during the initial join process. Under ideal condition, the system guarantees range-based lookups within each routing hub in $O\log^2 n/k$ when each node maintains k fixed links to the other nodes.

Note that, while the notion of a circular overlay is similar to DHTs, Mercury do not use any randomizing cryptographic hash functions for placing the nodes and data on the overlay. In contrast, Mercury overlay network is organized based on set of links. These links include the:

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

- Successor and predecessor links within the local attribute hub;
- k links to other nodes in the local attribute hub (intrahub links);
- One link per hub (interhub link) that aids in communicating with other attribute hubs and resolving multiattribute range queries.

Note that k intrahub links is a configurable parameter and could be different for different nodes in the attribute overlay. In this case the total routing table size at a node is $k + 2$. When a node n_k is presented with a message to find a node that maintains a range value $[l_i, r_i]$, it chooses the neighbor n_i such that the clockwise distance $d(l_i, v)$ is minimized; in this case the node n_i maintains the attribute range value $[l_i, r_i]$. The key to the message routing performance of Mercury is the choice of k intrahub links. To set up each link i, a node draws a number $x \in I$ using the harmonic probability distribution function:

$$p_n(x) = \frac{1}{n \log x}.$$

Following this, a node n_i attempts to add the node n' in its routing table that manages the attribute range value $r + (M_a - m_a) \times x$, where m_a and M_a are the minimum and maximum values for attribute a. For routing a data record D, the system routes to the value $\pi_a(D)$. For query Q, $\pi_a(Q)$ is a range. In this case first the message is routed to the first node that holds the starting range values; then the range contiguity property is used to spread the query along the overlay network.

PHT: Prefix Hash Tree — The work in [54] presents a mechanism for implementing range queries over a DHT-based system via a trie-based scheme. The bucket in the trie is stored at the DHT node obtained by hashing its corresponding prefixes. The resulting data structure is referred as a trie.[2] In the PHT, every vertex corresponds to a distinct prefix of the data domain being indexed. The prefixes of the nodes in the PHT form a universal prefix set.[3] The scheme associates a prefix label with each vertex of the tree. Given a vertex with label l, its left and right child vertices are labeled l_0 and l_1, respectively. The root of the tree is always labeled with the attribute name, and all subsequent vertexes are labeled recursively.

A data item is mapped and stored at the node having the longest prefix match with the node label. A node can store up to B items; if this threshold is exceeded, a node is recursively divided into two child nodes. Hence, this suggests that data items are only stored in the leaf nodes in the PHT, and the PHT itself grows dynamically based on distribution of inserted values. This logical PHT is distributed across nodes in the DHT-based network. Using the DHT lookup operation, a PHT node with label l is thus assigned to a node with the identifier closest to HASH(l). Lookup for a range query in a PHT network is performed by locating the node corresponding to the longest common prefix in the range. When such a node is found, parallel traversal of its subtree is done to retrieve all the desired items. Note that significant query lookup speedup can be achieved by dividing the range into a number of subranges.

JXTA: JXTA Search — JXTA Search [71] is an open framework based on the JXTA [105] routing substrate. A JXTA search network consists of search hubs, information providers, and information consumers. The network message communication protocol is based on the XML format. In the JXTA network search hubs are organized into N distinct groups. These groups are referred to as *advertisement groups*. These search hubs act as a point of contact for providers and consumers. Furthermore, each search hub is a member of a network of hubs that has at least one representative of hubs from every advertisement group. These groups are termed *query groups*. Hence, in this case there is 100 percent reachability to all stored information in the network.

Every information provider in the network registers its resource information with its local search hub. Each hub periodically sends an update message (new additions and deletions of registrations) to all the hubs in its advertisement group. If the grouping of hubs is content-based, the advertisement is forwarded to the relevant representative for that content. Whenever an information consumer wishes to look for data on the search network, it issues an information request query to the hub it knows or in which it has membership. The hub that receives this query first searches its local index and then other hubs in its advertisement group. If a match is found in the same advertisement group, the query is forwarded to that hub. If the query cannot be resolved in the local advertisement group, it is broadcast to all remaining advertisement groups using query group membership information. However, if the search network is organized based on content, the query is routed to the advertisement group responsible for indexing the desired content.

P2PR-Tree: An R-Tree-Based Spatial Index for P2P Environments — The work in [106] presents a scheme for adopting the R-tree [108] in a P2P setting. A P2PR-tree statically divides the d-dimensional attribute space (universe) into a set of blocks (rectangular tiles). The blocks formed as a result of initial division of the space forms level 0 of the distributed tree. Furthermore, each block is statically divided into a set of groups, which constitute level 1 in the tree. Any further division on the group level (and subsequently of the subgroup) is done dynamically and designated as subgroups at level i ($i \geq 2$). When a new peer joins the system, it contacts one of the existing peers, which informs it about the minimum bounding rectangle (MBR) of the blocks. Using this overall block structure information, a peer decides to which block(s) it belongs.

When relevant block(s) are determined, a peer queries other peers in the same block for compiling group-related MBR information. It also queries at least one peer in every other group. Using this group structure information, a peer knows about its own group. After determining the group, the same process is utilized for determining the subgroups and so on. Effectively, a peer maintains the following routing information:

- Pointers to all blocks in the universe;
- Pointers to all groups in its block;
- Pointers to all subgroups in its group;
- Finally, pointers to all peers in its subgroup;

The scheme defines a threshold value on the maximum number of peers in a group and subgroup denoted G_{Max} and SG_{Max}, respectively.

A query Q_L for an object is propagated recursively top down starting from level 0. When a query arrives at any peer

[2] *A trie is a multiway retrieval tree used for storing strings over an alphabet in which there is one node for every common prefix, and all nodes that share a common prefix hang off the node corresponding to the common prefix.*

[3] *A set of prefix is a universal prefix set if and only if for any infinite binary sequence* b *there is exactly one element in the set that is a prefix of* b.

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

P_i in the system, P_i checks whether its MBR covers the region indexed by the query. If so, P_i searches its own R-tree and returns the results, and the search is terminated at that point. Otherwise, the peer forwards the query to the relevant block, group, subgroup, or peer using its routing table pointers. This process is repeated until the query block is located or the query reaches a dead end of the tree.

Comparison of Surveyed Techniques: Scalability and Load Balancing

A majority of the surveyed approaches utilize a logical index structure that distributes the data among peers in a decentralized GRIS. The logical structure maintains a *d*-dimensional ($d \geq 1$) index space over the DHT key space, and forms the basis for the routing and indexing of data objects. Some approaches (Table 4) support only 1*d* queries for every distinct routing space. MAAN, Pub/Sub-1, and Pub/Sub-2 utilize variants of the SHA-1 hashing scheme for range partitioning 1*d* data over the DHT key space. We call these approaches variants of SHA-1, as they create a logical index space over the DHT key space, which is utilized by the query routing heuristics. These algorithms did not consider the case of data skew that can lead to routing load imbalance among the peers.

P-tree and Adaptive proposed a distributed version of the $B-+$ tree index as the basis for range partitioning 1*d* data. The PHT approach uses a trie-based structure for enabling 1*d* range queries in a peer-to-peer network. XenoSearch organizes resource information in the form of a logical tree where the leaves are the individual XenoServers. Query routing in XenoSearch is based on aggregation points (APs). An AP is managed by a XenoServer node in the system and is responsible for all the query computation for ranges of values covered by the AP. The Pastry IDs for the XenoServer responsible for an AP can be computed algorithmically. An AP owner in the system is similar to a super-peer, which is responsible for handling all query computation intersecting its region of ownership. The Adaptive approach considered the case of data skew and proposed a solution based on the load bbalancing matrix (LBM); PHT, P-tree, and XenoSearch did not propose any solution to this problem.

HPProtocol uses inverse Hilbert mapping to map the 1*d* index space to CAN's *d*-dimensional key space. Mercury directly operates on the attribute space along with a random sampling technique utilized for facilitating query routing and load balancing. A serious limitation of all the above approaches is the message overhead involved in maintaining a separate routing space for each attribute dimension. Furthermore, searching in a *d*-dimensional space requires querying every dimension separately and then finding an intersection. This leads to high message communication overhead for lookup and update queries. Clearly, these are not scalable ways to organize a grid resource attribute data set that has many dimensions.

The JXTA system does not create a logical index space over the distributed search network — instead, search is based on query broadcast among the advertisement group. This might prove costly in terms of number of messages generated. The Sword and Dgrid systems use a variant of SHA-1 hashing that partitions the DHT key space among different attribute types. Both Sword and Dgrid systems store all the attribute values in a single DHT ring. The Sword query resolution scheme is similar to MAAN, so it is also costly in terms of routing hops and messages generated. The AdeepGrid approach encodes all the resource attributes into a single object and then performs SHA-1 hashing to generate a Pastry ring identifier. However, in this case the authors do not address the issue of data skew. Furthermore, the proposed search techniques are not capable of returning deterministic results in all cases.

There are also some approaches that have utilized spatial indices for distributing the data among peers (refer to Table 5). Spatial indices, including Hilbert curves [21], Z-curves [102], k-d tree [1028], MX-CIF quadtree [99], R-tree [106], and R*-tree [104], have the capability to logically organize a *d*-dimensional index space over a single DHT key space. SFC-based indices, including Hilbert curves and Z-curves, have issues with routing load balance in case of a skewed index distribution. However, as the authors point out, SFC index load can be balanced through external techniques. In the case of Hilbert curves, dynamic techniques such as node virtualization and load partitioning with neighbor peers are utilized for this purpose. In the XenoSearch-II system Hilbert curves are utilized for mapping the *d*-dimensional index space to the 1*d* key space of Chord. However, XenoSearch-II does not propose any technique to counter load imbalance among peers.

The indexing approach based on Z-curves required an external load balancing technique. In the same work they introduced a P2P version of a k-d tree. This approach also has routing load-balance issues that need to be addressed. In another recent work, an MX-CIF quadtree-based spatial index has been proposed. DragonFly utilizes an index similar to the MX-CIF quadtree with the difference that it does not allow recursive decomposition of index space. Instead, the index cells are split as they exceed the preconfigured load threshold value (similar to Meghdoot). The authors argue that their approach does not require explicit load balancing algorithms in contrast to that of the others. The P2P-based R*-tree index in [104] uses CAN as the routing space. The index space is partitioned among super-peers and passive peers. The bulk of the query load is handled by the super-peers in the network similar to the Gnutella [39] system.

Meghdoot does not utilize any spatial index for organizing a *d*-dimensional data set. Instead, it utilizes a basic 2*d* CAN space for indexing a *d*-dimensional data set. Furthermore, Meghdoot incorporates a dynamic technique to counter the data skew issue. The load balancing technique in Meghdoot splits an overloaded index cell (zone) among lightly loaded peers. The P2P R-tree index divides the *d*-dimensional attribute space into a set of blocks (similar to the MX-CIF quadtree index); these blocks form the root of the distributed index tree. The work also includes a dynamic load division technique in case a peer index cell gets overloaded. However, this is an early work, and does not provide any bounds on messages and routing hops required in a *d*-dimensional index search.

To summarize, spatial indices are better suited to handling the complexity of grid resource queries than 1*d* data indices (as proposed in P-tree, MAAN, XenoSearch, etc.). However, even spatial indices have routing load balance issues in case of a skewed data set. Nevertheless, they are more scalable in terms of the number of hops and messages generated while searching in a *d*-dimensional space.

Security and Trust Issues in Peer-to-Peer Systems

The peer-to-peer nature of a distributed system raises serious challenges in the domains of security and trust management. Implementing a secure decentralized grid system requires solutions that can efficiently facilitate the following: preserve the privacy of participants, ensure authenticity of the partici-

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

pants, robust authorization, securely route messages between distributed services, and minimize loss to the system due to malicious participants.

The privacy of the participants can be ensured through secret key-based symmetric cryptographic algorithms such as 3DES, RC4 etc. These secret keys must be securely generated and distributed in the system. Existing key management systems such as public key algorithms (including DH, RSA, elliptic) and Kerberos (trusted third party) can be utilized for this purpose. Authentication of the participants can be achieved through trust enforcement mechanisms such as X.509 certificates (public key infrastructure) [127], Kerberos (third party authentication), distributed trust, and SSH. Authentication based on X.509 certificates warrants a trusted certifying authority (CA) in the system.

A grid participant presents an X.509 certificate along with an associated private key (the combination of these entities forms a system-wide unique credential) in order to authenticate itself with a remote service. A system can have a single CA that is trusted by all the participants. However, the single CA approach has limited scalability. An alternative to this is to have multiple CAs combining together to form a trust chain. In this case a certificate signed by any CA in the system has global validity. The GSI [128] implementation of PKI supports dynamic trust chain creation through the Community Authorization Service (CAS) [129]. This is based on the policy that two participants bearing proxy certificates signed by the same user will inherently trust each other. A Kerberos-based implementation has significant shortcomings as it requires synchronous communication with the ticket granting server in order to set up communication between a client and server. If the ticket granting server goes offline or has a security breach, there is no way the system can operate. In an X.509-based implementation, a CA can certify the credentials offline.

Having said that, a majority of implementations do rely on centralized trust enforcement entities such as a CA or a ticket granting authority. The JXTA [130] system provides a completely decentralized X.509-based PKI. Each JXTA peer is its own CA and issues a certificate for each service it offers. Peer CA certificates are distributed as part of the service advertisement process. Each of the CA certificate is verified via the *Poblano: "web of trust,"* a distributed reputation management system. A similar distributed trust mechanism called PeerReview [131] has also been proposed. These distributed trust management systems deter malicious participants through behavioral auditing. An auditor node A checks if it agrees with the past actions of an auditee node B. In case of disagreement, A broadcasts an accusation of B. Interested third party nodes verify evidence and take punitive action against the auditor or auditee.

The SSH-based authentication scheme is comparatively easier to implement as it does not require trusted third party certification. However, it does not allow the creation of a dynamic trust chain, and if a participant's private key is compromised, it requires every public key holder to be informed of this event. PlanetLab utilizes SSH-based authentication wherein the centralized PlanetLab Central service is responsible for distribution or copying of the keys. Unlike X.509 and Kerberos implementation, SSH does not support certificate translation (i.e., from X.509 to Kerberos or vice versa). Transport layer security protocols such as TLS [132] and SSL [133] are used for message encryption and integrity checking as they are transported from one host to the other on the Internet.

Authorization deals with the verification of an action a participant is allowed to undertake after a successful authentication. In a grid site owners have the privilege to control how their resources are shared among participants The resource sharing policy takes into account the participant's identity and membership in groups or virtual organizations. Globus-based grid installation defines the access control list using a Gridmap file. This file simply maintains a list of the distinguished names of the grid users and the equivalent local user account names to which they are to be mapped. Access control to a resource is then left up to the local operating system and application access control mechanisms.

Implementing a secure and trusted routing [134] primitive requires a solution to the following problems: secure generation and assignment of node IDs, securely maintaining the integrity of routing tables, and secure message transmission between peers. Secure node ID assignment ensures that an attacker or a malicious peer cannot choose the value of node IDs that can give it membership in the overlay. If the node assignment process is not secure, an attacker could sniff into the overlay with a chosen node ID and get control over the local objects, or influence all traffic to and from the victim node. The node ID assignment process is secured by delegating this capability to a central trusted authority. A set of trusted CAs are given the capability to assign node IDs to peers and sign node ID certificates, which bind a random node ID to the public key that uniquely identifies a peer and an IP address. The CAs ensure that node IDs are chosen randomly from the ID space, and prevent nodes from forging node IDs. Furthermore, these certificates give the overlay a public key infrastructure, suitable for establishing encrypted and authenticated channels between nodes. Secure message forwarding on the Internet can be achieved through secure transport layer connections such as TLS and SSL.

RECOMMENDATIONS

The surveyed DHT-based index services provide the basic platform for organizing and maintaining a decentralized grid resource discovery system. A grid system designer should follow a layered approach such as OPeN [99] in architecting and implementing a resource discovery system. The OPeN architecture consists of three layers: the *application*, *core services*, and *connectivity* layers. The application layer implements all the logic that encapsulates the query requirements of the underlying grid computing environment (the computational grids, data grids, etc.). The core services layer undertakes the tasks related to consistency management of virtual d-dimensional indices. The connectivity layer provides services related to key-based routing, overlay management, and replica placement. The application service, in conjunction with the core services, undertakes the resource discovery tasks including distributed information updates, lookups, and virtual index consistency management. The management of application and ccore services layers can be delegated to a component of broker software. We refer to this broker component as a *grid peer* service. Maintenance of the connectivity layer can be left to the basic DHT implementations such as FreePastry[4] and OpenDHT [135]. For further information, interested readers can refer to one of our recent works, such as [136], which utilizes a spatial publish/subscribe index [98] to facilitate a decentralized grid resource discovery system.

We recommend to grid system developers that for implementing the core services layer they utilize the spatial indices surveyed in this article. Overall, spatial indices are superior to $1d$ indices as they incur fewer messages for d-dimensional object lookups and updates. Although there are different

[4] *FreePastry is an open source implementation of Pastry.*

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

trade-offs involved with each of the spatial indices, basically they can all support scalability and grid resource indexing. A given spatial index would perform optimally in one scenario, but the performance could degrade if the data distribution changed significantly.

Open Issues

Peer-to-peer-based organization of grid resource discovery services promises an attractive and efficient solution to overcome the current limitations associated with the centralized and hierarchical model. However, the P2P nature of the system raises other serious challenges, including security [137], trust, reputation, and interoperational ability between distributed services. Enforcing trust among peers (a component of grid broker service) that host the indexing services warrants robust models for managing a peer's reputation and secure communication. A majority of the current solutions for security and trust management rely on centralized trust management entities such as CAs and ticket granting authorities. Achieving a completely decentralized security infrastructure is certainly a challenging future research direction. Recent efforts in this direction include emergence of distributed trust management systems such as PeerReview and Poblano. However, these trust management systems rely on behavioral auditing of the participant, and the distributed auditing process can take a while until a malicious participant is identified and shunted out of the system. This delay can allow ample opportunity for the malicious participant to effect significant harm on the system.

The current models of distributed systems, including grid computing and P2P, computing suffer from a knowledge and resource fragmentation problem. By knowledge fragmentation, we mean that various research groups in both academia and industry work in an independent manner. They define standards without any proper coordination. They give very little attention to the interoperatibility between related systems. Such disparity can be seen in the operation of various grid systems including Condor-G, Nimrod-G, OurGrid, Grid-Federation, Tycoon, and Bellagio. These systems define independent interfaces, communication protocols, superscheduling, and resource allocation methodologies. In this case users have access to only those resources that can understand the underlying grid system protocol. Hence, this leads to the distributed resource fragmentation problem.

A possible solution to this can be federating these grid systems based on universally agreed standards (similar to the TCP/IP model that governs the current Internet). The core to the operation and interoperability of the Internet component is the common resource indexing system, DNS. Both the grid and P2P communities clearly lack any such global or widely accepted service. These systems do not expose any API or interfaces that can help them interoperate. In recent times we have seen some efforts toward developing a generic grid service-oriented architecture, more commonly referred to as open grid service architecture (OGSA). Core grid developers also define common standards through the GGF. The Web service resource framework (WSRF) defines a new set of specifications for realizing the OGSA vision of grid and Web services. The WSRF can overcome the cross-platform interoperational ability issues in grid computing. However, it still cannot glue the gaps between various grid systems because of the basic differences in interfaces, communication protocols, superscheduling, and resource allocation methodologies.

Possible solutions to overcome knowledge and resource fragmentation include:

- Availability of a robust, distributed, scalable resource indexing/organization system;
- Evolution of common standards for resource allocation and application superscheduling;
- Agreement on using common middleware for managing grid resources such as clusters and SMPs;
- Defining common interfaces and APIs that can help different related systems to interoperate and coordinate activities.

Summary and Conclusion

In the recent past we have observed an increase in the complexity involved with grid resources, including their management policies, organization, and scale. Key elements that differentiate a computational grid system from a PDCS include:

- Autonomy;
- Decentralized ownership;
- Heterogeneity in management policies, resource types, and network interconnect;
- Dynamicity in resource conditions and availability.

Traditional grid systems [12, 14, 138] based on centralized information services are proving to be a bottleneck with regard to scalability, fault tolerance, and mechanism design. To address this, P2P-based resource organization is being advocated. P2P organization is scalable, adaptable to dynamic network conditions, and highly available.

In this work we present a detailed taxonomy that characterizes issues involved in designing a P2P/decentralized GRIS. We classify the taxonomies into two sections: resource taxonomy and P2P taxonomy. Our resource taxonomy highlights the attributes related to a computational grid resource. Furthermore, we summarize different kinds of queries that are being used in current computational grid systems. In general, grid superscheduling query falls under the category of d-dimensional point or window query. However, it still remains to be seen whether a universal grid resource query composition language is required to express different kinds of grid RLQs and RUQs.

We present classification of P2P approaches based on three dimensions: P2P network organization, approaches to distribution of data among peers, and routing of d-dimensional queries. In principle, a data distribution mechanism directly dictates how a query is routed among relevant peers. A d-dimensional resource index is distributed among peers by utilizing data structures such as SFCs, quadtrees, R-trees, and Kd-trees. Some of the approaches have also modified existing hashing schemes to facilitate the one-dimensional range queries in a DHT network. Every approach has its own merits and limitations. Some of these issues are highlighted in the resource and P2P network organization taxonomy section.

Acknowledgments

First, we should like to acknowledge the authors of the papers whose work has been surveyed and utilized in developing the taxonomy in this article. We thank our group members at the University of Melbourne — Marcos Assuncao, Al-Mukaddim Khan Pathan, Xingchen Chu and Md Mustafizur Rahman — for their comments. We should also like to thank Adriana Iamnitchi (assistant professor, computer science and engineering, University of South Florida), Jon Crowcroft (professor of communications systems in the Computer Laboratory at the University of Cambridge), David Spence (CCLRC Rutherford Appleton Laboratory, e-Science Center, Oxford), and Manish

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

Parashar (professor of electrical and computer engineering, Rutgers University) for their valuable feedback on the initial version of the article. Special thanks to Dr. Tejal Shah, Krishna Nandiminti, and Hussein Gibbins for proofreading the article. Both Krishna and Hussein work as research programmers in the GRIDS Laboratory, University of Melbourne. We are also grateful to the Department of Education, Science and Training (DEST), Australian Government, and the University of Melbourne for funding the Ph.D. candidature of Rajiv Ranjan. This work is also supported through an Australian Research Council discovery project grant. We would also like to thank the anonymous reviewers for their constructive comments that have helped us in improving the quality of our presentation substantially.

References

[1] D.S. Milojicic *et al.*, "Peer-to-Peer Computing," Tech. rep. HPL-2002-57, HP Labs, 2002.
[2] N. F. Noy, "Semantic Integration: A Survey of Ontology-Based Approaches," *SIGMOD Records*, vol. 33, no. 4, 2004, pp. 65–70.
[3] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Commun.*,vol. 8, no. 4, 2001, pp. 10–17.
[4] D. Barbara, "Mobile Computing and Databases-A Survey," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, 1999, pp. 108–17.
[5] G. H. Forman and J. Zahorjan, *The Challenges of Mobile Computing*, vol. 27, no. 4, 1994, pp. 38–47.
[6] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998.
[7] C. S. Yeo *et al.*, "Utility Computing on Global Grids, Handbook of Computer Networks," H. Bidgoli, Ed., Wiley, Apr. 2006.
[8] J. M. Schopf, "Ten Actions When Superscheduling," Global Grid Forum, 2001.
[9] R. Ranjan, R. Buyya, and A. Harwood, "A Case for Cooperative and Incentive Based Coupling of Distributed Clusters," *Proc. 7th IEEE Int'l Conf. Cluster Computing*, Boston, MA, 2005.
[10] N. Andrade *et al.*, "OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing," *Proc. 9th Wksp. Job Scheduling Strategies for Parallel Processing*, LNCS, Springer.
[11] H. Shan, L. Oliker, and R. Biswas, "Job Superscheduler Architecture and Performance in Computational Grid Environments," *Proc. ACM/IEEE Conf. Supercomputing*, 2003, p. 44.
[12] D. Abramson, R. Buyya, and J. Giddy, "A Computational Economy for Grid Computing and Its Implementation in the Nimrod-G Resource Broker," *Future Generation Comp. Sys. J.*, vol. 18, no. 8, Oct., 2002, pp. 1061–74.
[13] S. Venugopal, R. Buyya, and L. Winton, "A Grid Service Broker for Scheduling Distributed e-Science Applications on Global Data Grids," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 6, 2006, pp. 685–99.
[14] J. Frey *et al.*, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Proc. 10th IEEE Int'l Symp. High Performance Distrib. Computing*, 2001, pp. 237–46.
[15] J. Yu and R. Buyya, "A Novel Architecture for Realizing Grid Workflow Using Tuple Spaces," *Proc. 5th IEEE/ACM Grid Wksp.*, 2004, pp. 119–28.
[16] T. Fahringer *et al.*, "Askalon: A Tool Set for Cluster and Grid Computing," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2–4, pp. 143–69, 2005.
[17] M. Cai *et al.*, "Maan: A Multi-Atribute Addressable Network for Grid Information Services," *Proc. 4th IEEE/ACM Int'l Wksp. Grid Computing*, 2003, pp. 184–91.
[18] A. Iamnitchi and I. Foster, *A Peer-to-Peer Approach to Resource Location in Grid Environments*, 2004, pp. 413–29.
[19] K. Czajkowski *et al.*, "Grid Information Services for Distributed Resource Sharing," *Proc. 10th IEEE Int'l Symp. High Performance Distrib. Computin*, 2001, pp. 181.
[20] J. Yu, S. Venugopal, and R. Buyya, "Grid Market Directory: A Web and Web Services Based Grid Service Publication Directory," *J. Supercomputing*, vol. 36, no. 1, 2006, pp. 17–31.
[21] C. Schmidt and M. Parashar, "Flexible Information Discovery in Decentralized Distributed Systems," *Proc. 12th Int'l Symp. High Performance Distrib. Computing*, 2003.
[22] A. Andrzejak and Z. Xu, "Scalable, Efficient Range Queries for Grid Information Services," *Proc. 2nd IEEE Int'l Conf. Peer-to-Peer Computing*, 2002.
[23] D. Ouelhadj *et al.*, "A Multi-Agent Infrastructure and a Service Level Agreement Negotiation Protocol for Robust Scheduling in Grid Computing," *Proc. European Grid Conf.*, LNCS, 2005.
[24] K. Czajkowski, I. Foster, and C. Kesselman, "Agreement-Based Resource Management," *Proc. IEEE*, vol. 93, no. 3, Mar. 2005..
[25] C. Courcoubetis and V. Siris, "Managing and Pricing Service Level Agreements for Differentiated Services," *Proc. 6th IEEE/IFIP Int'l Conf. QoS*, London, U.K., May–June 1999.
[26] R. Ranjan, A. Harwood, and R. Buyya, "A SLA-Based Coordinated Superscheduling Scheme and Performance for Computational Grids," Tech. rep., GRID S-TR-2006-8, Grid Comp. and Distrib. Sys. Lab., Univ. of Melbourne, Australia, 2006.
[27] J. Litzkow *et al.*, "Condor — A Hunter of Idle Workstations," *Proc. 8th IEEE Int'l Conf. Distrib. Computing Sys.*, 1988.
[28] B. Bode *et al.*, "PBS: The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters," *Proc. 4th Linux Showcase and Conf.*, Atlanta, GA, Oct. 2000.
[29] W. Gentzsch, "Sun Grid Engine: Towards Creating a Compute Power Grid," *Proc. 1st IEEE Int'l Symp. Cluster Computing and the Grid*, Brisbane, Australia, 2001.
[30] S. Chapin, J. Karpovich, and A. Grimshaw, "The Legion Resource Management System," *Proc. 5th Wksp. Job Scheduling Strategies for Parallel Processing*, San Juan, P.R., Apr. 1999.
[31] A. Luther *et al.*, "Peer-to-Peer Grid Computing and a NET-Based Alchemi Framework," *High Performance Computing: Paradigm and Infrastructure*, L. Yang and M. Guo, Eds., Wiley, 2004.
[32] S. Zhou, "LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems," *Proc. Wksp. Cluster Computing*, Tallahassee, FL, 1992.
[33] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," *Software Practice and Experience*, vol. 32, no. 2, 2002, pp. 135–64.
[34] K. Lai, B. A. Huberman, and L. Fine, "Tycoon: A Distributed Market-Based Resource Allocation System," Tech. rep., HP Labs, 2004.
[35] X. Zhang, J. L. Freschl, and J. M. Schopf, "A Performance Study of Monitoring and Information Services for Distributed Systems," *Proc. 12th Int'l Conf. High Performance Distrib. Computing*, June 2003.
[36] S. Zanikolas and R. Sakellariou, "A Taxonomy of Grid Monitoring Systems," *Future Generation Comp. Sys. J.*, vol. 21, no. 1, Jan. 2005, pp. 163–88.
[37] S. Fitzgerald *et al.*, "A Directory Service for Configuring High-Performance Distributed Computations," *Proc. 6th IEEE Symp. High Performance Distrib. Computing*, 1997, pp. 365–75.
[38] F. D. Sacerdoti *et al.*, "Wide Area Cluster Monitoring with Ganglia," *Proc. 5th IEEE Int'l Conf. Cluster Computing*, Tsim Sha Tsui, Kowloon, Hong Kong, 2003.
[39] Y. Chawathe *et al.*, "Making gnutella-Like P2P Systems Scalable," *SIGCOMM '03,* 2003, pp. 407–18.
[40] I. Stoica *et al.*, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *SIGCOMM '01,* 2001, pp. 149–60.
[41] S. Ratnasamy *et al.*, "A Scalable Content-Addressable Network," *SIGCOMM '01*, 2001, pp. 161–72.
[42] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Proc. IFIP/ACM Int'l Conf. Distrib. Sys. Platforms*, Heidelberg, Germany, 2001, pp. 329–59.
[43] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," Tech. rep. UCB/CSD-01-1141, UC Berkeley, Apr. 2001.
[44] R. Huebsch *et al.*, "Querying the Internet with Pier," *Proc. 19th Int'l Conf. Very Large Databases*, Berlin, Germany, Sept. 2003.
[45] M. Castro *et al.*, "Scribe: A Large-Scale and Decentralized Application Level Multicast Infrastructure," *IEEE JSAC*, vol. 20, no. 8, 2002, pp. 1489–99.
[46] A. Mislove *et al.*, "Experiences in Building and Operating Epost, a Reliable Peer-to-Peer Application," *Proc. EuroSys Conf.*, Leuven, Belgium, 2006, pp. 147–59.

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

[47] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery," *Proc. 1st Int'l Conf. Pervasive Computing*, 2002, pp. 195–210.
[48] A. S. Cheema, M. Muhammad, and I. Gupta, "Peer-to-Peer Discovery of Computational Resources for Grid Applications," *Proc. 6th IEEE/ACM Int'l Wksp. Grid Computing*, 2005.
[49] Y. M. Teo, V. Mar., and X. Wang, "A DHT-Based Grid Resource Indexing and Discovery Scheme," Singapore-MIT Alliance Annual Symp., 2005.
[50] D. Oppenheimer *et al.*, "Design and Implementation Trade-Offs for Wide-Area Resource Discovery," *Proc. 14th IEEE Symp. High Performance*, Research Triangle Park, NC, July, 2005.
[51] F. Dabek *et al.*, "Wide-Area Cooperative Storage with CFS," *Proc. 18th ACM Symp. Op. Sys. Principles*, Banff, Alberta, 2001, pp. 202–15.
[52] A. Crainiceanu *et al.*, "Querying Peer-to-Peer Networks Using P-Trees," *Proc. 7th Int'l Wksp. Web and Databases*,2004, pp. 25–30.
[53] A. R. Bharambe, M. Agrawal, and S. Seshan. "Mercury: Supporting Scalable Multi-Attribute Range Queries," *SIGCOMM '04*, Portland, OR, 2004, pp. 353–66.
[54] S. Ramabhadran *et al.*, "Brief Announcement: Prefix Hash Tree," *Proc. ACM PODC '04*, St. Johns, Canada, 2004.
[55] D. Spence and T. Harris, "Xenosearch: Distributed Resource Discovery in the Xenoserver Open Platform," *Proc. 12th IEEE Int'l Conf. High Performance Distrib. Computing*, 2003, p. 216.
[56] F. Bonnassieux, R. Harakaly, and P. Primet, "MapCenter: An Open Grid Status Visualization Tool," *Proc. ICSA 15th Int'l Conf. Parallel and Distrib. Computing Sys.*, Louisville, KY, 2002.
[57] S. Andreozzi *et al.*, "GridICE: A Monitoring Service for Grid Systems," *Future Generation Comp. Sys.*, vol. 21, no. 4, 2005, pp. 559–71.
[58] R. L. Ribler, "Autopilot: Adaptive Control of Distributed Applications," *Proc. 7th Int'l Symp. High Performance Distrib. Computing*, 1998, pp. 172–79.
[59] W. Smith. A System for Monitoring and Management of Computational Grids," *Proc. 31st IEEE Int'l Conf. Parallel Processing*, 2002.
[60] M. Baker and G. Smith, "GridRM: A Resource Monitoring Architecture," *Proc. 3rd Int'l Wksp. Grid Computing*, LNCS, Springer, 2002, pp. 268–73.
[61] P. Stelling *et al.*, "A Fault Detection Service for Wide Area Distributed Computations," *Proc. 7th IEEE Int'l Conf. High Performance Distrib. Computing*, 1998, p. 268.
[62] Z. Balaton *et al.*, "From Cluster Monitoring to Grid Monitoring Based on GRM," *Proc. 7th Int'l Euro-Par Conf.*, 2001, pp. 874–81.
[63] D. Gunter *et al.*, "Netlogger: A Toolkit for Distributed System Performance Analysis," *Proc. 8th IEEE Int'l Symp. Modeling, Analysis and Simulation of Comp. and Telecommun. Sys.*, 2000, pp. 267.
[64] R. Wolski, N. Spring, and C. Peterson, "Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service," *Supercomputing '97*, 1997, pp. 1–19.
[65] R. Wismuller, J. Trinitis, and T. Ludwig, "OCM-A Monitoring System for Interoperable Tools," *Proc. SIGMETRICS Symp. Parallel and Distrib. Tools*, 1998, pp. 1–9.
[66] P. A. Dinda *et al.*, "The Architecture of the Remos System," *Proc. 10th IEEE Int'l Symp. High Performance Distrib. Computing*, 2001, p. 252.
[67] H. L. Truong and T. Fahringer, "SCALEA-G: A Unified Monitoring and Performance Analysis System for the Grid," *LNCS*, 2004, pp. 202–11.
[68] H.B. Newman *et al.*, "MonALISA: A Distributed Monitoring Service Architecture," 2003.
[69] B. P. Miller *et al.*, "The Paradyn Parallel Performance Measurement Tool," *IEEE Computer*, vol. 28, no. 11, 1995, pp. 37–46.
[70] B. Tierney *et al.*, "A Grid Monitoring Architecture," Global Grid Forum, 2002.
[71] S. Waterhouse *et al.*, "Distributed Search in P2P Networks," *IEEE Internet Computing*, vol. 06, no. 1, 2002, pp. 68–72.
[72] I. Clarke *et al.*, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Proc. Int'l Wksp. Designing Privacy Enhancing Technologies*, 2001, pp. 46–66.
[73] S. Saroiu, P. Gummadi, and S. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. 19th ACM Symp. Op. Sys. Principles*, 2003.
[74] G. Pallis and A. Vakali, "Insight and Perspectives for Content Delivery Networks," *Commun. ACM*, vol. 49, no. 1, 2006, pp. 101–06.
[75] A. Raza Butt, R. Zhang, and Y. C. Hu, "A Self-Organizng Flock of Condors," *Proc. 2003 ACM/IEEE Conf. Supercomputing*, 2003.
[76] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int'l J. Supercomputer Apps.*, vol. 11, no. 2, 1997, pp. 115–28.
[77] P. Druschel and A. Rowstron, "Storage Management and Caching in Past, A Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proc. 18th ACM Symp. Op. Sys. Principles*, Banff, Alberta, Canada, 2001, pp. 188–201.
[78] J. Kubiatowicz *et al.*, "Oceanstore: An Architecture for Global-Scale Persistent Storage," *SIGPLAN Notes*, vol. 35, no. 11, 2000, pp. 190–201.
[79] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," *Proc. 7th IEEE Int'l Symp. High Performance Distrib. Computing*, 1998, p. 140.
[80] R. L. Rivest, "Partial Match Retrieval Algorithms," *SIAM J. Computing*, vol. 5, no. 1, 1976, pp. 19–50.
[81] K. A. Berman and J. L. Paul, *Fundamentals of Sequential and Parallel Algorithms*, PWS, 1997.
[82] T. H. Cormen *et al.*, *Introduction to Algorithms*, 2nd ed., MIT Press, 2001.
[83] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman, "Scalability Issues in Large Peer-to-Peer Networks — A Case Study of Gnutella," Tech. rep., Univ. Cincinnati, 2001.
[84] K. Calvert, M. Doar, and E. W. Zegura, "Modeling Internet Topology," *IEEE Commun. Mag.*, June 1997.
[85] Q. Lv *et al.*, "Search and Replication in Unstructured Peer-to-Peer Networks," *Proc. 16th Int'l Conf. Supercomputing*, 2002, pp. 84–95.
[86] H. Balakrishnan *et al.*, "Looking Up Data in P2P Systems," *Commun. ACM*, vol. 46, no. 2, 2003, pp. 43–8.
[87] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk, "Cryptographic Hash Functions: A Survey, Citeseer," ist.psu.edu/bakhtiari95 cryptographic.html, 1995.
[88] D. Karger *et al.*, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," *Proc. 29th Ann. ACM Symp. Theory of Computing*, El Paso, TX, 1997, pp. 654–63.
[89] B. Preneel, "The State of Cryptographic Hash Functions," *Lectures on Data Security, Modern Cryptology in Theory and Practice*, Summer School, Aarhus, Denmark, July 1998, 1999, pp. 158–82.
[90] K. Lua *et al.*, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Commun. Surveys and Tutorials*, 2005.
[91] R. Ranjan, A. Harwood, and R. Buyya, "A Taxonomy of Peer-to-Peer Based Complex Queries: A Grid Perspective," http://arxiv.org/abs/cs/0610163, 2006.
[92] J. Li *et al.*, "Comparing the Performance of Distributed Hash Tables Under Churn," *Proc. 3rd Int'l Wksp. Peer-to-Peer Sys.*, Feb. 2004.
[93] M. Castro, M. Costa, and A. Rowstron, "Should We Build Gnutella On A Structured Overlay?" *SIGCOMM Comp. Commun. Rev.*, vol. 34 no. 1, 2004, pp. 131–36.
[94] P. Linga *et al.*, "Building An Efficient and Stable P2P DHT Through Increased Memory and Background Overhead," *Proc. 2nd Int'l Wksp. Peer-to-Peer Sys.*, 2003.
[95] A. Gupta, B. Liskov, and R. Rodrigues, "One Hop Lookups for Peer-to-Peer Overlays," *Proc. 9th Wksp. Hot Topics in Op. Sys.*, Lihue, HI, May, 2003, pp. 7–12.
[96] D. Spence *et al.*, "Location Based Placement of Whole Distributed Systems," *Proc. ACM Conf. Emerging Network Experiment and Tech.*, Toulouse, France, 2005, pp. 124–34.
[97] J. Gao and P. Steenkiste, "An Adaptive Protocol for Efficient Support of Range Queries in DHT-Based Systems," *Proc. 12th IEEE Int'l Conf. Network Protocols*, 2004, pp. 239–50.
[98] L. Chan and S. Karunasekera, "Designing Configurable Publish-Subscribe Scheme for Decentralized Overlay Networks," *AINA'07, Proc. IEEE 21st Int'l Conf. Advanced Info. Networking and Apps.*, May 2007.
[99] E. Tanin, A. Harwood, and H. Samet. A Distributed Quadtree Index for Peer-to-Peer Settings," *Proc. Int'l Conf. Data Eng.*, 2005, pp. 254–55.

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

[100] P. Triantafillou and I. Aekaterinidis, "Content-Based Publish/-Subscribe Over Structured P2P Networks, " *Proc. 1st Int'l Wksp. Discrete Event-Based Systems*, 2004.
[101] D. Tam, R. Azimi, and H.A. Jacobsen, "Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables," *Proc. Int'l Wksp. Databases, Info. Sys. and Peer-to-Peer Computing*, 2003.
[102] P. Ganesan, B. Yang, and H. Garcia-Molina, "One Torus to Rule Them All: Multi-Dimensional Queries in P2P Systems," *Proc. 7th Int'l Wksp. Web and Databases*, Paris, France, 2004, pp. 19–24.
[103] A. Gupta *et al.*, "Meghdoot: Content-Based Publish/Subscribe Over P2P Networks," *Proc. 5th ACM/IFIP/USENIX Int'l. Conf. Middleware*, 2004, pp. 254–73.
[104] B. Liu, W. Lee, and D. L. Lee, "Supporting Complex Multi-Dimensional Queries in P2P Systems," *Proc. 25th IEEE Int'l Conf. Distrib. Computing Sys.*, Columbus, OH, 2005, pp. 155–64.
[105] L. Gong, "JXTA: A Network Programming Environment," *IEEE Internet Computing*, IEEE Computer Society, Los Alamitos, CA, vol. 05, no. 3, 2001, pp. 88–95.
[106] A. Mondal, Y. Lifu, and M. Kitsuregawa, "P2PR-Tree: An r-tree-based Spatial Index for Peer-to-Peer Environments," *Proc. Int'l Wksp. Peer-to-Peer Computing and Databases* (held in conjunction with EDBT), Springer-Verlag, 2004, pp. 516–25.
[107] S. Berchtold, C. Bohm, and Hans-Peter Kriegal, "The Pyramid-Technique: Towards Breaking the Curse of Imensionality," *Proc. 1998 ACM SIGMOD Int'l Conf. Mgmt. of Data*, Seattle, WA, 1998, pp. 142–53.
[108] V. Gaede and O. Gunther, "Multidimensional Access Methods," *ACM Computing Surveys*, vol. 30, no. 2, 1998, pp. 170–231.
[109] M. Bienkowski, M. Korzeniowski, and F. M. auf der Heide, "Dynamic Load Balancing in Distributed Hash Tables," *4th Int'l. Wksp. Peer-to-Peer Sys.*, 2005, pp. 217–25.
[110] A. Rao *et al.*, "Load Balancing in Structured P2P Systems," *Proc. 2nd Int'l Wksp. Peer-to-Peer Sys.*, 2003.
[111] T. Asano *et al.*, "Space-Filling Curves and Their Use in the Design of Geometric Data Structures," *Theoretical Comp. Sci.*, vol. 181, no. 1, 1997, pp. 3–15.
[112] H. V. Jagadish, "Linear Clustering of Objects with Multiple Attributes," *Proc. ACM Int'l Conf. Mgmt. of Data*, Atlantic City, NJ, 1990, pp. 332–42.
[113] J. Orenstein. A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces," *Proc. ACM Int'l Conf. Mgmt. of Data*, Atlantic City, NJ, 1990, pp. 343–52.
[114] H. V. Jagadish, "Analysis of the Hilbert Curve for Representing Two-Dimensional Space," *Info. Processing Lett.*, vol. 62, no. 1, 1997, pp. 17–22.
[115] F. Korn, B. Pagel, and C. Faloutsos, "On the 'Dimensionality Curse' and the 'Self-Similarity Blessing'," *IEEE Trans. Knowledge and Data Eng.*, vol. 13, no. 1, 2001, pp. 96–11.
[116] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1989.
[117] M. J. Berger and S. H. Bokhari, "A Partitioning Strategy for Non-Uniform Problems on Multiprocessors," 1987.
[118] P. Ganesan, M. Bawa, and H. Garcia-Molina, "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems," Tech. rep., Stanford Univ., 2004.
[119] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Commun. ACM*, vol. 18, no. 9, 1975, pp. 509–17.
[120] S. Hand *et al.*, "Controlling the Xenoserver Open Platform," *IEEE Conf. Open Architectures and Network Programming*, 2003, pp. 3–11.
[121] D. Spence, "An Implementation of a Coordinate Based Location System," Tech. rep., Univ. of Cambridge Comp. Lab., 2003.
[122] J. Aspnes and G. Shah, "Skip Graphs," *Proc. 14th Annual ACM-SIAM Symp. Discrete Algorithms*, Baltimore, MD, 2003, pp. 384–93.
[123] O. D. Sahin *et al.*, "A Peer-to-Peer Framework for Caching Range Queries," *Proc. 20th Int'l Conf. Data Eng.*, 2004, p. 165.
[124] N. Beckmann *et al.*, "The r*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. 1990 ACM SIGMOD Int'l Conf. Mgmt. of Data*, 1990, pp. 322–31.
[125] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network," *Proc. 19th IEEE Int'l Conf. Data Eng.*, 2003, p. 49.
[126] S. Rhea *et al.*, "OpenDHT: A Public DHT Service and Its Uses," *SIGCOMM '05*, 2005, pp. 73–84.
[127] R. Housley *et al.*, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," 2002.
[128] V.Welch *et al.*, "Security for Grid Services," *Proc. 12th IEEE Int'l Conf. High Performance Distrib. Computing*, 2003, pp. 48–57.
[129] L. Pearlman *et al.*, "A Community Authorization Service for Group Collaboration," *Proc. 3rd Int'l Wksp. Policies for Distrib. Sys. and Networks*, 2002, pp. 50.
[130] W. Yeager and J. Williams, "Secure Peer-to-Peer Networking: The JXTA Example," *IT Professional*, vol. 4, no. 2, 2002, pp. 53–57.
[131] P. Durschel, "The Renaissance of Decentralized Systems," keynote talk, 15th IEEE Int'l Symp. High Performance Distrib. Computing, Paris, France, 2006.
[132] P. Chown, "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)," 2002.
[133] W. Chou, "Inside SSL: Accelerating Secure Transactions," *IT Professional*, vol. 4, no. 5, 2002, pp. 37–41.
[134] M. Castro *et al.*, "Secure Routing for Structured Peer-to-Peer Overlay Networks," *Op. Sys. Rev.*, vol. 36, no. SI, 2002, pp. 299–314.
[135] S. Ratnasamy, I. Stoica, and S. Shenker, "Routing Algorithms for DHTS: Some Open Questions," *1st Int'l. Wksp. Peer-to-Peer Sys.*, 2002, pp. 45–52.
[136] R. Ranjan, "A Scalable, Robust, and Decentralized Resource Discovery Service for Large Scale Federated Grids," Tech. rep. GRIDS-TR-2007-6, Grids Lab., CSSE Department, Univ. of Melbourne, Australia, 2007.
[137] E. Sit and R. Morris, "Security Considerations for Peer-to-Peer Distributed Hash Tables," *1st Int'l Wksp. Peer-to-Peer Sys.*, 2002, pp. 261–69.
[138] A. Auyoung *et al.*, "Resource Allocation in Federated Distributed Computing Infrastructures," *Proc. 1st Wksp. Op. Sys. and Architectural Support for the On-Demand IT Infrastructure*, Boston, MA, Oct. 2004.

BIOGRAPHIES

RAJIV RANJAN (rranjan@csse.unimelb.edu.au) is a final-year Ph.D. student in the Peer-to-Peer and Grids Laboratory at the University of Melbourne, Australia. Prior to beginning work on his Ph.D., he completed a Bachelor's degree securing first rank in the Computer Engineering Department (North Gujarat University, India) in 2002. He has worked as a research assistant (honors project) at the Physical Research Laboratory (a unit of the Department of Space, Government of India), Ahmedabad, Gujarat, India. He was also a lecturer in the Computer Engineering Department of Gujarat University, where he taught undergraduate computer engineering courses including systems software, parallel computation, and advance operating systems. His current research interest lies in the algorithmic aspects of resource allocation and resource discovery in decentralized grid and peer-to-peer computing systems. He has served as a reviewer for journals including *Future Generation Computer Systems*, *Journal of Parallel and Distributed Computing*, *IEEE Internet Computing*, and *IEEE Transactions on Computer Systems*. He has also served as an external reviewer for conferences including IEEE Peer-to-Peer Computing ('04, '05, '06), IEEE/ACM Grid Computing '06, and Parallel and Distributed Computing, Applications and Technologies '07).

AARON HARWOOD (aharwood@csse.unimelb.edu.au) completed his Ph.D. degree at Griffith University on high-performance interconnection networks in 2002. During that time he worked on several software projects including the development of a VLSI layout package and integrated circuit fabrication virtual laboratory now in use for classroom instruction. He has also worked at Research Institute for Industrial Science and Technology (RIST), South Korea, on computer simulation for a robot traffic light controller system. He then joined the University of Melbourne as a lecturer in the Department of Computer Science and Software Engineering where his research focused on the topological properties and software engineering of peer-to-peer systems for high-performance computing. In 2003 he co-founded the Peer-to-Peer Networks and Applications Research Group (www.cs.mu.oz.au/p2p), of which he is now acting director. He recently developed one of

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.

the first parallel computing platforms for peer-to-peer networks. He is a program committee member for the 6th IEEE/ACM Workshop on Grid Computing.

RAJKUMAR BUYYA (raj@csse.unimelb.edu.au) is a senior lecturer and director of the Grid Computing and Distributed Systems Laboratory within the Department of Computer Science and Software Engineering at the University of Melbourne. He received his B.E and M.E in computer science and engineering from Mysore and Bangalore Universities in 1992 and 1995, respectively; and his Ph.D. in computer science and software engineering from Monash University, Melbourne, Australia, in April 2002. He was awarded the Dharma Ratnakara Memorial Trust Gold Medal in 1992 for his academic excellence at the University of Mysore, India. He received Leadership and Service Excellence Awards from the IEEE/ACM International Conference on High Performance Computing in 2000 and 2003. He has authored/co-authored over 130 publications. He has co-authored three books: *Microprocessor x86 Programming* (BPB Press, 1995), *Mastering C++* (Tata McGraw Hill, 1997), and *Design of PARAS Microkernel*. Books on emerging topics he has edited include *High Performance Cluster Computing* (Prentice Hall, 1999) and *High Performance Mass Storage and Parallel I/O* (IEEE and Wiley, 2001). He has also edited proceedings of 10 international conferences and served as guest editor for major research journals. He is serving as an Associate Editor of *Future Generation Computer Systems: The International Journal of Grid Computing: Theory, Methods and Application*. He served as a speaker with the IEEE Computer Society Chapter Tutorials Program (1999–2001) and Founding Co-Chair of the IEEE Task Force on Cluster Computing (TFCC), 1999–2004, and Interim Co-Chair of the IEEE Technical Committee on Scalable Computing (TCSC), 2004–September 2005, and a member of Executive Committee of the IEEE Technical Committee on Parallel Processing (TCPP), 2004–2005. He is currently serving as Elected Chair of the TCSC.

Authorized licensed use limited to: UNIVERSITY OF MELBOURNE. Downloaded on October 24, 2008 at 20:21 from IEEE Xplore. Restrictions apply.