A Study on Peer-to-Peer Based Discovery of Grid Resource Information

Rajiv Ranjan, Aaron Harwood and Rajkumar Buyya
P2P Networks Group and GRIDS Laboratory
Department of Computer Science and Software Engineering
University of Melbourne, Victoria, Australia
{rranjan,aharwood,raj}@csse.unimelb.edu.au

December 1, 2006

Abstract

Efficient Resource discovery mechanism is one of the fundamental requirement for Grid computing systems, as it aids in resource management and scheduling of applications. Resource discovery activity involve 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 the centralised and hierarchical information server approach. However, both of these approaches have serious limitations in regards to *scalability, fault-tolerance and network congestion*. To overcome these limitations, indexing resource information using a decentralised (such as Peer-to-Peer (P2P)) network model has been actively proposed in the past few years.

This article investigates various decentralised resource discovery techniques primarily driven by P2P network model. To summarise, this article presents a: (i) summary of current state of art in grid resource discovery; (ii) resource taxonomy with focus on computational grid paradigm; (iii) P2P taxonomy with focus on extending the current structured systems (such as Distributed Hash Tables) for indexing *d-dimensional* grid resource queries; (iv) detailed survey of existing works that can support d-dimensional grid resource queries; and (v) 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.

1 Introduction

The last few years have seen the emergence of a new generation of distributed systems that scale over the Internet, operate under decentralised settings and are dynamic in their behavior (participants can leave or join the system). One such system is referred to as Grid Computing and other similar systems include P2P Computing [78], Semantic Web [82], Pervasive Computing [102] and Mobile Computing [14, 45]. Grid Computing [47] 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. etc.

Grids can be primarily classified [123] into various types, depending on 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 computational power of globally distributed computers (e.g., TeraGrid, ChinaGrid, and APACGrid). Data Grids emphasize on a global-scale management of data to provide data access, integration and processing through distributed data repositories (e.g., LHCGrid, GriPhyN). Application service (provisioning) Grids focus on providing access to remote applications, modules; libraries hosted on data centers or computational Grids (e.g., NetSolve and GridSolve). Interaction Grids focused on interaction and collaborative visualization between participants (e.g., AccessGrid). Knowledge Grids aim towards knowledge acquisition, processing, 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, service to end users as IT utilities on subscription basis and provides 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 with computational grid being at the bottom most layer while the utility grid being at the top most layer. A grid at higher-level utilizes the services of grids that operate at lower layers in the design. For example, a Data Grid utilizes the services of 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 the Computational Grids. Computational Grids enable aggregation of different types of compute resources including clusters, supercomputers, desktops. In general, compute resources have two types of attributes: (i) 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); and (ii) dynamic attributes such as processor utilization, physical memory utilization, free secondary memory size, current usage price and network bandwidth utilization.

1.1 The Superscheduling Process and Resource Indexing

The Grid superscheduling [104] 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 [90], MyGrid [3], NASA-Superscheduler [105], Nimrod-G [2], GridBus-Broker [118], Condor-G [48] and workflow engines [124, 43]. Fig.1 shows an abstract model of a decentralised 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: (i) identifying and analyzing user's job requirements; (ii) querying GRIS [25, 62, 38, 125, 103, 5] for locating resources that match the job requirements; (iii) coordinating and negotiating Service Level Agreement (SLA) [85, 39, 36, 92]; and (iv) job scheduling. Grid resources are managed by their local resource management systems (LRMSes) such as Condor [71], Portable Batch System (PBS) [22], Sun Grid Engine (SGE) [53], Legion [30], Alchemi [74] and Load Sharing Facility LSF [129]. The LRMSes manage job queues, initiate and monitor their execution.

Traditionally, superschedulers including Nimrod-G, Condor-G and Tycoon [69] used services of centralised information services (such as R-GMA [127], Hawkeye [126], GMD [125], MDS-1 [44]) to index resource information. Under centralised organisation, the superschedulers send resource queries to a centralised 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: (i) highly prone to a single point of failure; (ii) lacks scalability; (iii) high network communication cost at links leading to the information server (i.e. network bottleneck, congestion); and (iv) 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 centralised approaches, a hierarchical organisation of information services has been proposed in systems such as MDS-3 [61] and Ganglia [98]. MDS-3 organizes Virtual Organisation (VO) [47] specific information directories in a hierarchy. A VO includes a set of GPs that agree on common resource sharing policies. Every VO in 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 as the centralised approach such as one-point of failure, and does not scale well for a large number of users/providers.

1.2 Decentralised Resource Indexing

Recently, proposals for decentralizing a GRIS have gained significant momentum. The decentralization of GRIS can overcome the issues related to current centralised and hierarchical organisations. A distributed system configuration is considered as decentralised *"if none of the participants in the system is more important than others, in case one of the participant fails then it is neither more or less harmful to the system than caused by the failure of any other participant in the system"*. An early proposal for decentralizing Grid information services was made by Iamnitchi

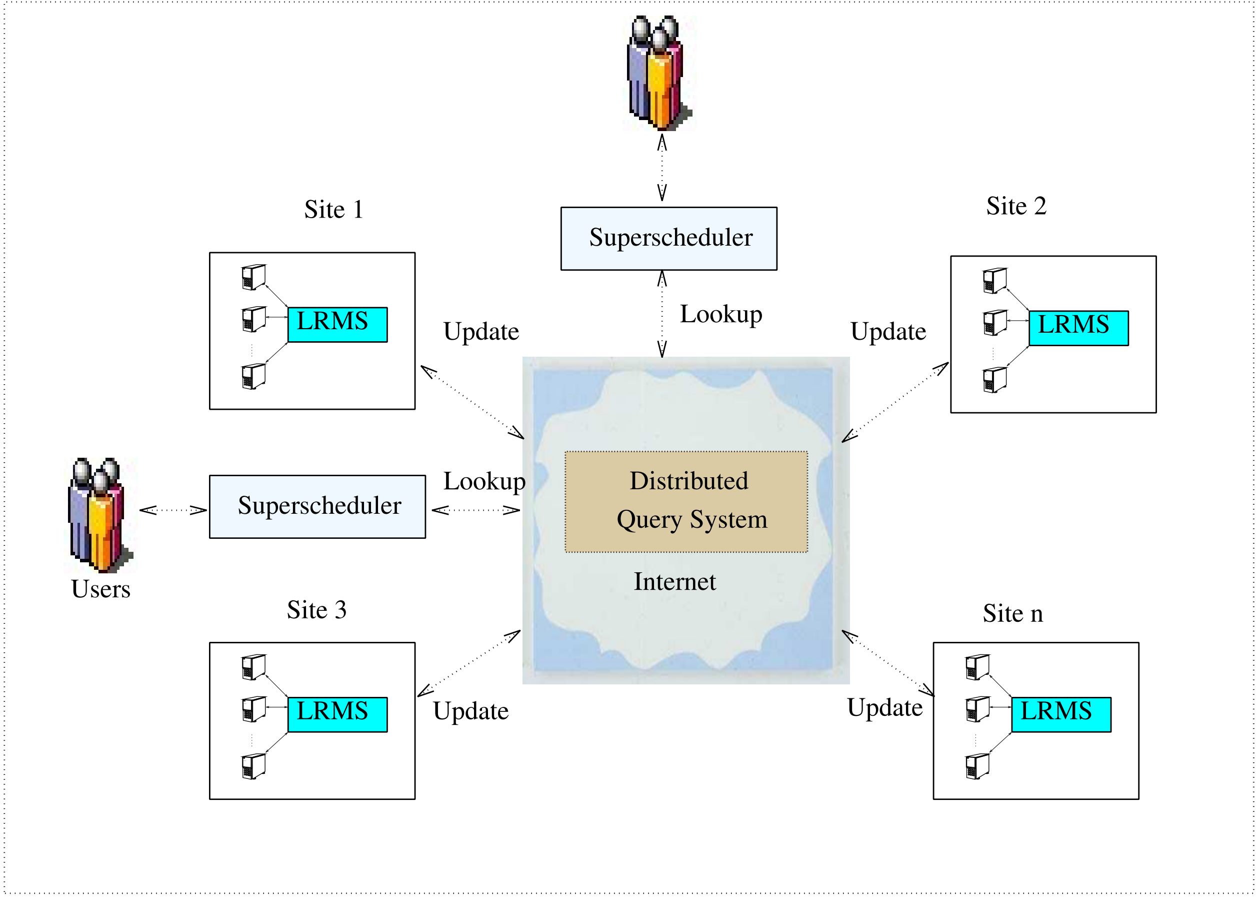

Figure 1: Superscheduling and resource queries

and Foster [62]. 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 look-up query which is forwarded in the P2P network using flooding (an approach similar to one applied in the unstructured P2P network Gnutella [31]. 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 can not 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 look-up systems including Chord [111], CAN [93], Pastry [97] and Tapestry [128] are primarily based on Distributed Hash Tables (DHTs). DHTs provide hash table like functionality at the Internet scale. A DHT is a data structure that associates a key with a data. Entries in the distributed hashtable are stored as a (key,data) pair. A data can be looked up within a logarithmic overlay routing hops if the corresponding key is known.

It is widely accepted that DHTs are the building blocks for next-generation large scale decentralised systems. Some of the example distributed systems that utilizes DHT routing substrate include distributed databases [59], group communication [28], E-mail services [79], resource discovery systems [13, 32, 114, 103, 83] and distributed storage systems [40]. Current implementations of DHTs are known to be efficient for 1-dimensional queries [59] 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 on a specific value. Current works including [32, 114, 103, 25, 37, 5, 83, 20, 88, 109] have studied and proposed different solutions to this problem.

1.3 Conceptual Design of a Distributed Resource Indexing System

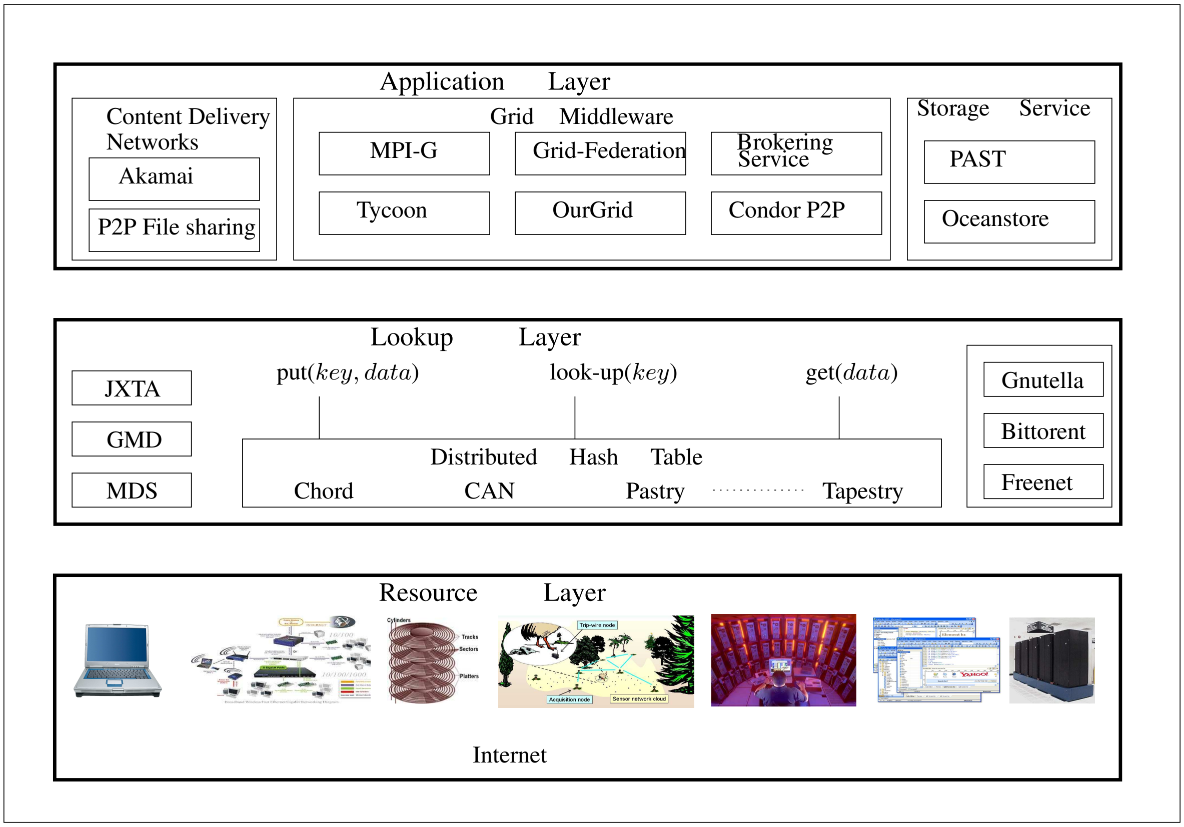

Figure 2: Distributed resource indexing: a layered approach

A layered architecture to build a distributed resource indexing system is shown in Fig. 2. The key components of a Internet-based resource indexing system includes:

- **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 (such as UNIX or Windows) and queuing systems (such as Condor, Alchemi, SGE, PBS,LSF).
- **Lookup layer:** This layer offers core services for indexing resources at the Internet scale. The main components at this layer are the middlewares that support Internet-wide resource look-ups. 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 [119], Grid Market Directory (GMD) [125] and unstructured P2P substrates such as Gnutella [31] and Freenet [34].
- **Application layer:** This layer includes the application services in various domains including: (i) Grid computing; (ii) distributed storage; (iii) P2P networks; and (iv) Content Delivery Networks (CDNs) [101], [86]. Grid computing systems including Condor-Flock P2P [24] uses 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 [46]

for resource indexing and superscheduling. The OurGrid superscheduling framework incorporates JXTA for enabling communication between OGPeers in the network. Distributed storage systems including PAST [42] and OceanStore [68] utilizes services of DHTs such as Pastry and Tapestry for resource indexing.

1.4 Paper organisation

The rest of the paper is organized as follows. Section 2 summarizes the approaches that are based on a non-P2P resource organisation model, specifically the centralised and hierarchical network models. Section 3 presents taxonomies related to general computational resources' attributes, look-up queries and organisation model. In section 4, we present taxonomies for P2P network organisation, d-dimensional data distribution mechanism and query routing mechanism. Section 5 summarizes various algorithms that model GRIS over a P2P network. Finally, we end this paper with discussion on open issues in section 6 and conclusion in section 7.

2 The State of Art in Grid Information Indexing

The work in [126] presents a comprehensive taxonomy on existing centralised and hierarchically organised GRISes. 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) [115] put forward by the Global Grid Forum (GGF). The main components of GMA are: (i) producer–daemon that monitors and publishes resource attributes to the registry; (ii) consumer–superschedulers that query the registry for resource information; (iii) registry–a service or a directory that allows publishing and indexing of resource information; (iv) republisher–any object that implements both producer and consumer functionality; and (v) schema repository–holds details such as type and schema about different kinds of events that are ambient in a GRIS. The work defines a scope-oriented taxonomy of existing GRIS. The systems are identified depending on the provision and characteristics of its producers and republishers.

Table 1: Summarizing centralised and hierarchical GRIS

Level 0	**Level 1**	**Level 2**	**Level 3**
MapCenter [23], GridICE [4]	Autopilot [95]	CODE [106], GridRM [9], Hawkeye [126], HBM [110], Mercury [12], NetLogger [55], NWS [121], OCM-G [120], Remos [41], SCALEA-G [117]	Ganglia [98], Globus MDS [38], MonALISA [81], Paradyn [77], RGMA [127]

- 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 an 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 [23] and GridICE [4] 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 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 [95] 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. The republisher of different functionality may be stacked upon each other but only in a predefined way. The only difference from Level 1 systems being the presence of a republisher in the system. Systems including GridRM [9], CODE [106] and Hawkeye are level 2 systems.
- Level 3 (Hierarchies of Republishers): This category of system allows for the hierarchical organisation of republishers in an arbitrary fashion. This functionality is not supported in the 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, Level 1 or Level 2 system. Systems such as MDS-3 [38] 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 (organisation of the republisher in a Level 2 system) of a GRIS. A republisher can be completely centralised, 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 the least supports event for hosts and network types.
- Stackable: denotes whether the concerned GRIS can work on top of another GRIS.

3 Resource Taxonomy

The taxonomy for a computational grid resource is divided into the following (refer to Fig. 3): (i) resource organisation; (ii) resource attribute; and (iii) resource query.

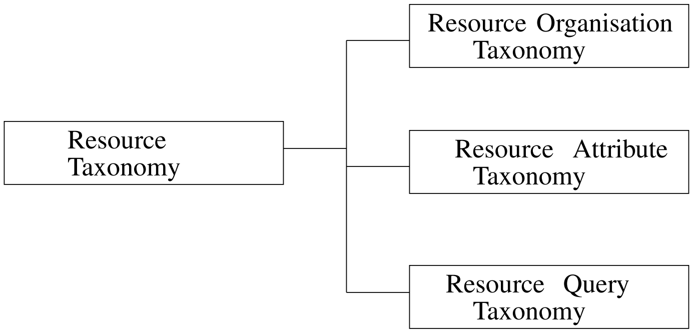

Figure 3: Resource taxonomy

3.1 Resource/GRIS organisation taxonomy

The taxonomy defines GRIS organisation as (refer to Fig. 4) :

- Centralised: Centralisation refers to the allocation of all query processing capability to single resource. The main characteristics of a centralised approach include control and efficiency. All look-up and update queries are sent to a single entity in the system. GRISes including RGMA [127] and GMD [125] are based on centralised organisation.
- Hierarchical: A hierarchical approach links GRIS's 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 system including MDS-3 [38] and Ganglia [98] are based on this network model.

- Decentralised: No centralised control, complete autonomy, authority and query processing capability is distributed over all resources in the system. The GRIS organized under this model is fault-tolerant, self-organizing and is scalable to large number of resources. More details on this organisation can be found in section 4.

  There are four fundamental challenges related to different organisation models including: (i) scalability; (ii) adaptability; (iii) availability; and (iv) manageability. Centralised models are easy to manage but do not scale well. When network links leading to the central server get congested or fail, then the performance suffers. Hence, this approach may not adapt well to dynamic network conditions. Further, it presents a single point of failure, so overall availability of the system degrades considerably. Hierarchical organisation overcomes some of these limitations including scalability, adaptability and availability. However, these advantages over a centralised model comes at the cost of overall system manageability. In this case, every site specific administrator has to periodically ensure the functionality of their local daemons. Further, the root node in the system may present a single point failure similar to the centralised model. Decentralised systems, including P2P, are coined as 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.

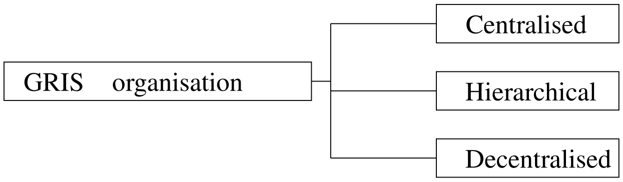

Figure 4: Resource organisation taxonomy

3.2 Resource Attribute Taxonomy

A compute grid resource is described by a set of attributes which is globally known to the application superschedulers. The superscheduler which is 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, compute resources have two types of attributes: (i) static or 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 (ii) dynamic or range valued attributes such as CPU utilisation, physical memory utilisation, free secondary memory size, current usage price and network bandwidth utilisation. Figure 5 depicts the resource attribute taxonomy.

3.3 Resource Query Taxonomy

The ability of superschedulers such as MyGrid, Grid-Federation Agent, , NASA-Superscheduler, Nimrod-G, Condor-Flock P2P to make effective application scheduling decision is directly governed by the efficiency of GRIS. Superschedulers need to query a GRIS to compile information about resource's utilisation, load and current access price for formulating the efficient schedules. Further, 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 pattern and accordingly set the price. The actual semantics of the resource query depends on the underlying Grid superscheduling model or Grid system model.

3.3.1 Resource Query Type

Superscheduling systems require two basic types of queries: (i) resource look-up query (RLQ) ; and (ii) 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

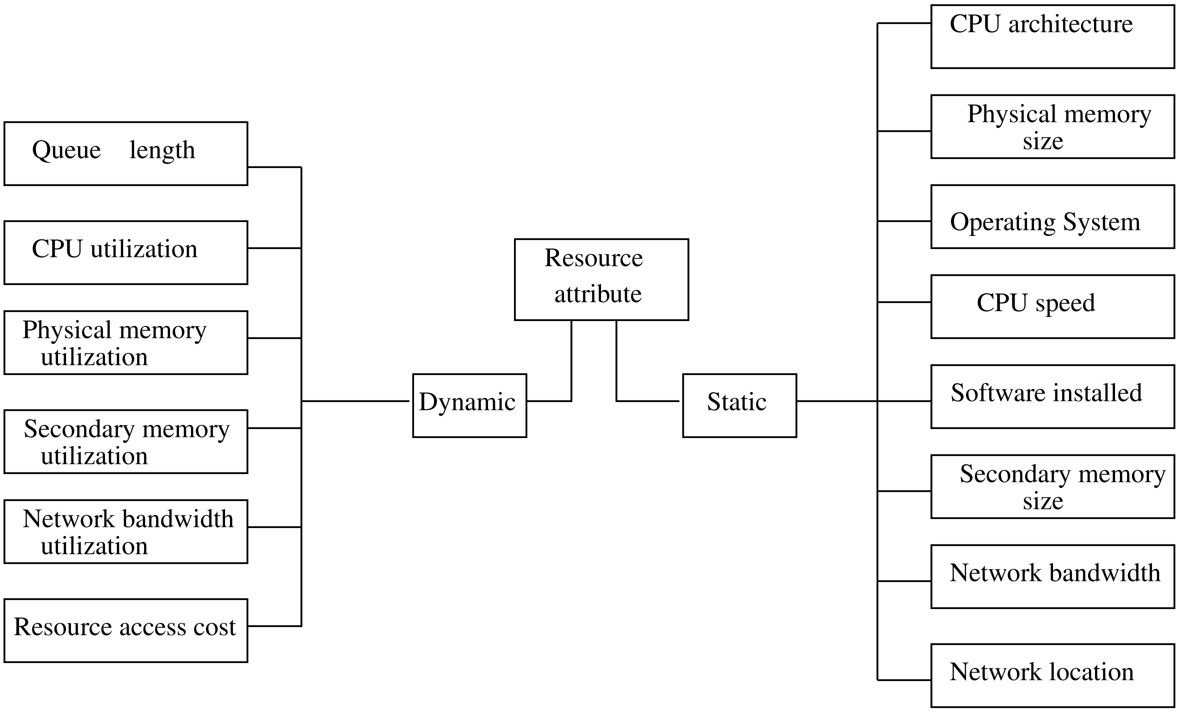

Figure 5: Resource attribute taxonomy

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. The distributed flocking is based on the P2P query mechanism. Once the job is migrated to the remote pool, basic matchmaking [89] mechanism is applied for resource allocation. In Table 2, we present RLQ and RUQ queries in some well-known superscheduling systems.

3.3.2 An Example Superscheduling Resource Query

In this section we briefly analyse the superscheduling query composition in the superscheduling system called Tycoon [69]. The Tycoon system applies market-based principles, in particular an auction mechanism, for resource management. Auctions are completely independent without any centralised control. Every resource owner in the system coordinates its own auction for local resources. The Tycoon system provides a centralised Service Location Service (SLS) for superschedulers to index resource auctioneers' information. Auctioneers register their status with the SLS every 30 seconds. If an auctioneer fails to update its information within 120 seconds then 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 requirement 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 centralised GRIS (referred to as service local services). 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 (such as CPU speed, memory size, disk space). An auctioneers RUQ has the following semantics:

$$total\ \ bids = 10\ \ \&\&\ \ CPU\ \ Arch = \text{``}pentium\text{``}\ \ \&\&\ \ CPU\ \ Speed = 2\ \ GHz\ \ \&\&\ \ Memory = 512$$

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 look-up query has the following semantics:

$$return\ \ auctioneers\ \ whose\ \ CPU\ \ Arch = \text{``}i686\text{``}\ \ \&\&\ \ CPU\ \ Speed \geq 1\ \ GHz\ \ \&\&\ \ Memory \geq 256$$

Table 2: Resource query in superscheduling systems

System Name	**Resource Lookup Query**	**Resource Update Query**	**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	Decentralised
Grid-Federation	Query decentralised 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 utilisation, memory, disk space, no. of free processors)	Decentralised
Nimrod-G	Query GMD or MDS for resources that matches jobs resource and QoS requirement	Update resource service price and resource type available	Centralised
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	Centralised
Our-Grid	MyPeer queries OGPeer for resources that match user's job requirements	Update network of favors credit for OurGrid sites in the community	Decentralised
Gridbus Broker	Query GMD or MDS for resources that matches jobs resource and QoS requirement	Update resource service price and resource type available	Centralised
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	Centralised
Bellagio	Query for resources based on CPU load, available memory, internode latency, physical and logical proximity	Update resource conditions including CPU , memory and network usage status	Decentralised
Mosix-Grid	Information available at each node through *gossiping algorithm*	Update CPU usage, current load, memory status and network status	Hierarchical

In Fig. 6, we present the taxonomy for GRIS RLQ and RUQ. In general, the queries [96] 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. Further, both the 1-dimensional and d-dimensional query can specify different kinds of constraints on the attribute values. If the query specifies a fixed value for each attribute then it is referred to as a *d-dimensional Point Query* (DPQ). However, in case the query specifies a range of values for attributes, then 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, Archi-

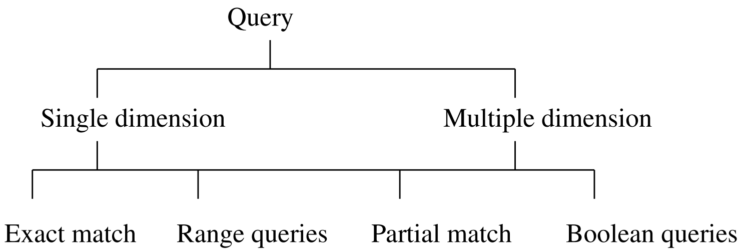

Figure 6: Resource query taxonomy

tecture='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' $\leq$ CPU-Speed $\leq$ '3 GHz' and '512MB' $\leq$ RAM $\leq$ '1 GB'. (Multiple Dimension Range Query).
- Boolean queries: All or some attribute values satisfying certain boolean conditions. Such as, ((not RAM $\leq$ '256 MB') and not No. of processors $\leq$ 5). (Multiple Dimension Boolean Query).

4 P2P Taxonomy

The taxonomy for P2P based GRIS is divided into the following (refer to Fig. 7): (i) P2P network organisation; (ii) data organisation; and (iii) d-dimensional query routing organisation.

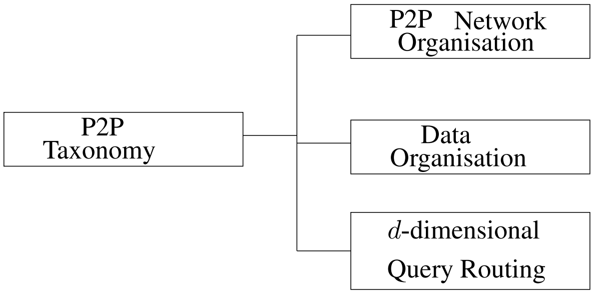

Figure 7: Peer-to-Peer network taxonomy

4.1 P2P Network Organisation

The network organisation refers to how peers are logically structured from the topological perspective. Fig. 8 shows the network organisation taxonomy of general P2P systems. Two categories are proposed in P2P literature [78]: unstructured and structured. An unstructured system is typically described by a power law random graph model [19, 35], 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 [65], [26] for unstructured systems can be found in [75]. Unstructured systems including Napster, Gnutella and Kazaa offer differing degrees of decentralization. The degree of decentralization refers to the extent peers can function

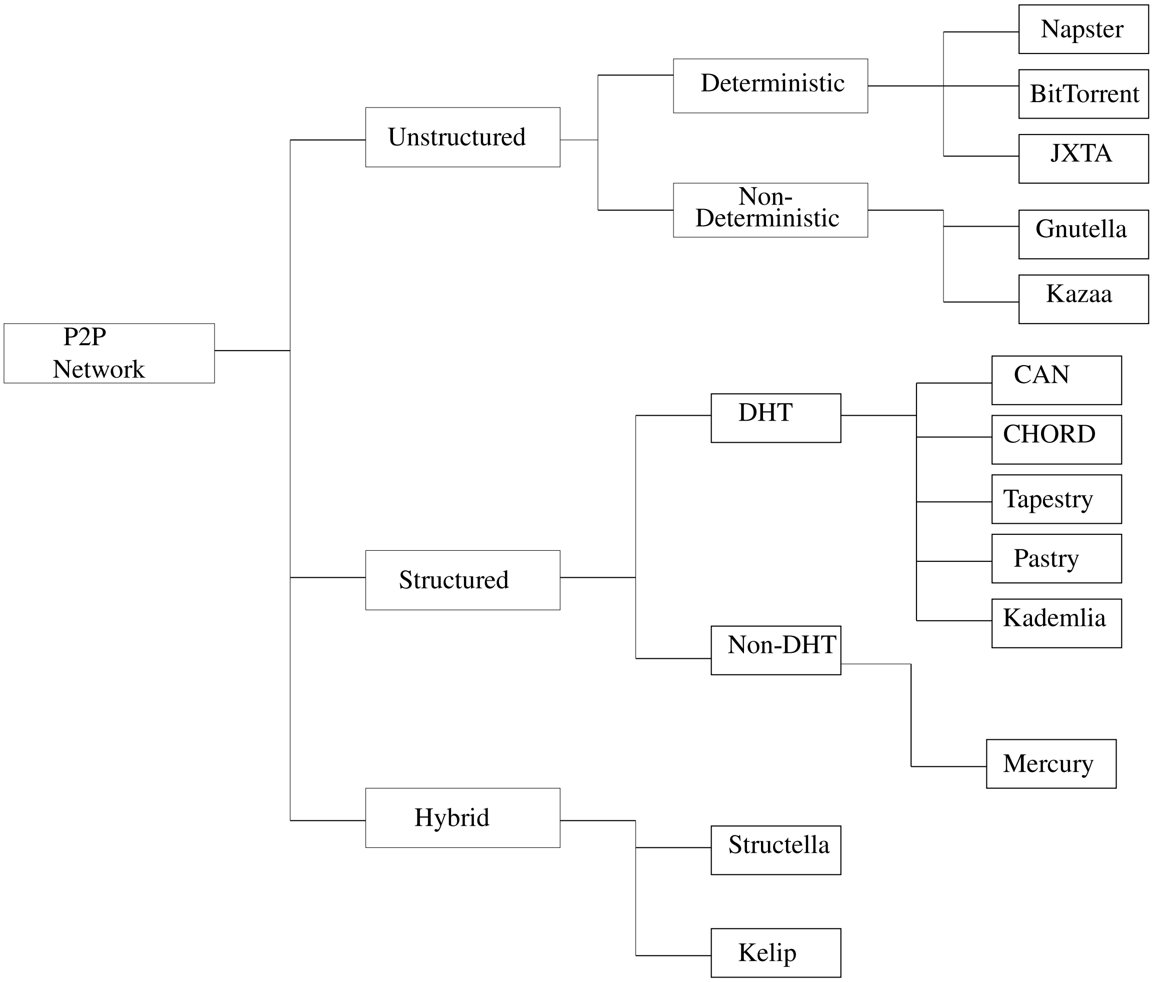

Figure 8: Peer-to-Peer network organisation taxonomy

independently with respect to efficient object look-up and query routing. Our taxonomy classifies unstructured systems as *deterministic* or *non-deterministic* [75].

Deterministic system means that a look-up operation will be successful within predefined bounds. Systems including Napster, BitTorrent fall into this category. In these systems, the object lookup operation is centralised while download is decentralised. Under centralised organisation, a specialised (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: (i) peers directly inform the server about the files they are currently holding (e.g. Napster); or (ii) by crawling the P2P network (an approach similar to a web search engine). The look up operations in these systems is deterministic and is 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 look-up and download operations are completely decentralised.

Other unstructured systems including Gnutella, Freenet, FastTrack and Kazaa offer non-deterministic query performance. Unlike Napster or BitTorrent, both object lookup and download operation in these systems are decentralised. Each peer maintains indexes for the objects it is currently holding. In other words, indexes are completely distributed. The Gnutella system employs a query flooding model for routing object queries. Every request for an object is flooded (broadcasted) to the directly connected peers, which in turn flood their neighboring peers. This approach is used in the GRIS model proposed by [62]. 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 non-scalability. 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 [11]: (i) generation of node-ids and object-ids, called keys, using cryptographic/randomizing hash functions such as SHA-1 [10, 66, 87]. 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; (ii) 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; and (iii) an efficient look-up query resolution scheme. Whenever a node in the overlay receives a look-up request, it must be able to resolve it within acceptable bounds such as in $O(\log(n))$ time. This is achieved by routing the look-up 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. Though at the core various DHTs (Chord, CAN, Pastry etc.) are similar, still there exists 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 [73, 91]. In Table-3, we present the comparative analysis of Chord, Pastry, CAN and Tapestry based on basic performance and organisation parameters. Comprehensive details about the performance of some common DHTs under churn can be found in [70].

Table 3: Summary of the complexity of structured P2P systems

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	$O(\log_b(n))$	$O(b\ \log_b(n)+b)$	$O(\log(n))$

Other classes of structured systems such as Mercury do not apply randomising hash functions for organising data items and nodes. The Mercury system organises nodes into a circular overlay and places data contiguously on this ring. As Mercury does not apply hash functions, data partitioning among nodes is non-uniform. Hence it requires an explicit load-balancing scheme. In recent developments, new generation P2P systems have evolved to combine both unstructured and structured P2P networks. We refer to this class of systems as hybrid. Structella [27] 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 design includes Kelips [60] and its variants. Nodes in 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. Other variant of Kelips [56] allows routing table entries to store information for every other node in the system. However, this approach is based on assumption that system experiences low churn rate [70]. Gossiping and one-hop routing approach has been used for maintaining the routing overlay in the work [108]. In Table 4, we summarize the different P2P routing substrate that are utilized by the existing algorithms for organizing a GRIS.

4.2 Data Organisation Taxonomy

Traditionally, DHTs have been efficient for 1-dimensional queries such as finding all resources that match the given attribute value. Extending DHTs to support DRQs, to index all resources whose attribute value overlap a given search space, is a complex problem. DRQs are based on ranges of values for attributes rather than on specific values. Compared to 1-dimensional queries, resolving DRQs is far more complicated, as there is no obvious total ordering of the points in the attribute space. Further, 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 [51] include efficient: (i) data distribution mechanisms; and (ii) data indexing or query routing techniques. In this section, we discuss various data distribution mechanisms while we analyse data indexing techniques in the next section.

A data distribution mechanism partitions the d-dimensional [17, 49] 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 [51]:

- 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 [21, 1].
- 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 (refer to section 5), d-dimensional data distribution mechanisms based on the following structures have been proposed (refer to Fig. 5): (i) space filling curves; (ii) tree-based structures; and (iii) variant of SHA-1/2 hashing. In Table 5, we summarise various data structures used in different algorithms for d-dimensional data distribution. Further, 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 data structure (*SFCs*) [6], [63] includes the Z-curve [84] and Hilbert's curve [64]. SFCs map the given d-dimensional attribute space into a 1-dimensional space. The work in [5] utilises space-filling curves (SFC), in particular the reverse Hilbert SFC for mapping a 1-dimensional attribute space to a two-dimensional CAN P2P space. Similarly, the work in [103] uses the Hilbert SFC to map a d-dimensional index space into a 1-dimensional space. The resulting 1-dimensional indexes are contiguously mapped on a Chord P2P network. The approach proposed in [51] utilises Z-curves for mapping d-dimensional space to 1-dimensional space. SFCs exhibit the locality property by mapping the points that are close in d-dimensional space to adjacent spaces in the 1-dimensional space. However, as the number of dimensions increases, locality becomes worse since SFCs suffer from "curse of dimensionality" [67]. Further, SFC based mapping fails to uniformly distribute the load among peers if the data distribution is skewed. Hence, this leads to a non-uniform query processing load for peers in the network.

Some of the recent works [113, 37, 52, 88] utilize tree-based data structures for organising the data. The approach proposed in [113] adopts the MX-CIF quadtree [100] 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, then recursive subdivision of the 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 to-tally different locations on the Chord space. Another recent work [29], uses the same base algorithm with an enhanced load balancing technique called recursive bisection [18]. 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 the load threshold. Hence, this approach has better load balancing properties as compared to the SFC-based approaches in the case of a skewed data set.

Other approaches including [114, 25] manipulate existing SHA-1/2 hashing for mapping d-dimensional data to the peers. MAAN addresses the 1-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 [116]. 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 in case 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.

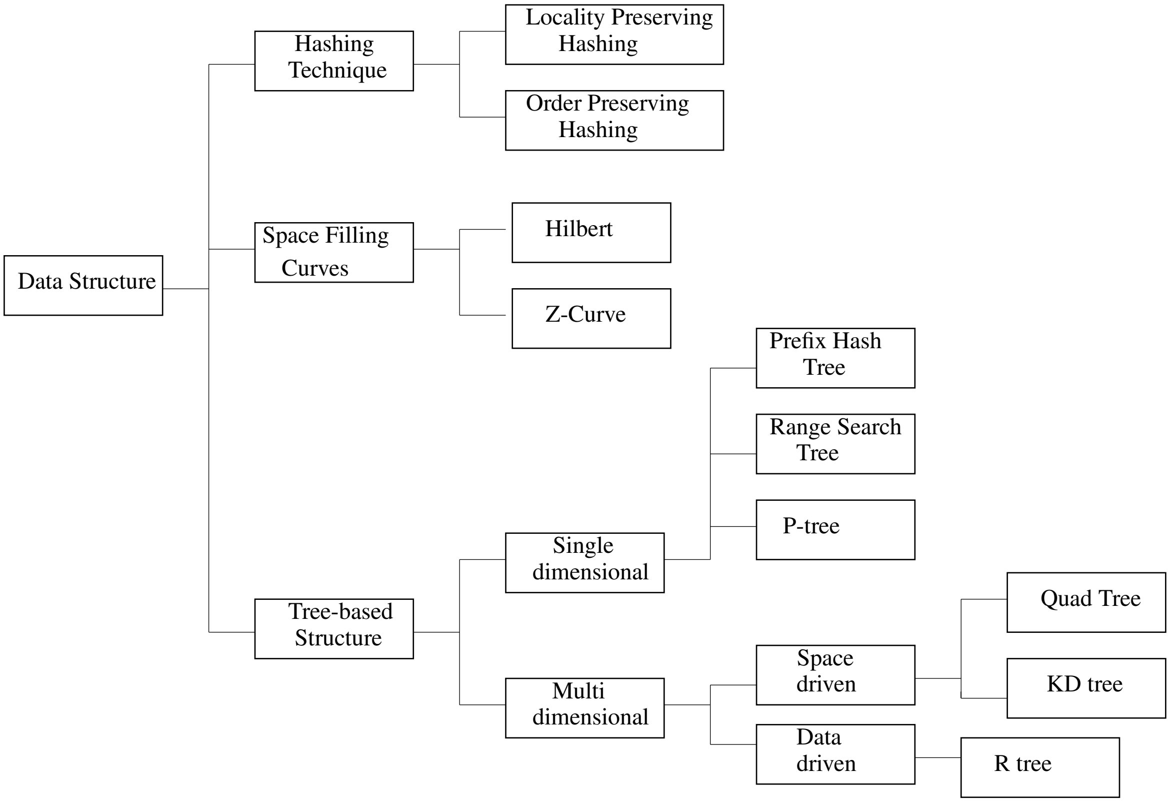

Figure 9: Data structure taxonomy

4.3 D-dimensional Query Routing Taxonomy

DHTs guarantee deterministic query lookup with logarithmic bounds on network message cost for 1-dimensional queries. However, Grid RLQs are normally DPQ or DRQ. Hence, existing routing techniques need to be augmented in order to efficiently resolve a DRQ. Various data structures that we discussed in previous section effectively create a logical d-dimensional index space over a DHT network. A look-up operation involves searching for a 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. Efficient query routing algorithm should exhibit the following characteristics [51]:

- Routing load balance: every peer in the network on the average should route forward/route approximately 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 look-up complexity involved with the existing algorithms.

Resolving a DRQ over a DHT network that utilises SFCs for data distribution consists of two basic steps [103]: (i) mapping the DRQ onto the set of relevant clusters of SFC-based index space; and (ii) routing the message to all peers that fall under the computed SFC-based index space. The simulation based study proposed in [51] 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 [50].

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 [113] introduced the concept of fundamental minimum level. This means that all the query processing and the data storage should start at that minimal level of the tree rather than at the root. Another approach [51] utilises a P2P version of a Kd-tree [16] for mapping d-dimensional data onto a CAN P2P space. The routing utilises the neighboring cells of the data structure. The nodes in this network that manage a dense region of space are likely to have large number of neighbors, hence leading to an unbalanced routing load.

Other approaches based on variants of standard hashing schemes (such as MAAN) apply different heuristics for resolving range queries. The single-attribute dominated query routing (SAQDR) heuristic abstracts resource attributes into two categories: (i) dominant attribute; and (ii) non-dominant attribute. 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 non-dominant attributes in the DRQ. The request is then forwarded to the next node which 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 the classification of the existing algorithms based on query resolution heuristic, and data locality preserving characteristics.

Table 4: Classification based on P2P routing substrate

Routing Substrate	**Network Organisation**	**Distributed Indexing Algorithm Name**
Chord	Structured	PHT [88], MAAN [25], Dgrid [114], Adaptive [52], DragonFly [29], QuadTree [113], Pub/Sub-2 [116], P-tree [37]
Pastry	Structured	XenoSearch [109], Adeep-Grid [32], Pub/Sub-1 [112]
CAN	Structured	HP-protocol [5], Squid [103], Kd-tree [51],Meghdoot [57],Z-curve [51], Super-P2P R*-Tree [72]
Bamboo	Structured	SWORD [83]
Epidemic-DHT [56]	Hybrid	XenoSearch-II [108]
Others	Unstructured	Mercury [20], JXTA search [54], P2PR-tree [80]

5 Survey of P2P based Grid Information Indexing

5.1 Pastry Based Approaches

5.1.1 Pub/Sub-1: Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables

The content-based Publish/Subscribe (Pub/Sub) system [112] is built using a DHT routing substrate. They use the topic-based Scribe [28] system which is implemented using Pastry [97]. The model defines different schema for publication and subscription messages for each application domain (such as 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,

Table 5: Classification based on data structure applied for enabling ranged search and look-up complexity

Algorithm Name	**Data Structure**	**Lookup Complexity**
PHT [88]	Trie	$O(\log\|D\|)$; D is the total number of bits in the binary string representation, for 1-dimensional range query
MAAN [25]	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 [114]	SHA-1 hashing	$O(\log_2 Y)$ for each dimension, Y is the total resource type in the system
SWORD [83]	N.A.	N.A.
JXTA search [54]	RDBMS	N.A.
DragonFly [29]	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 [113]	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
Pub/Sub-2 [116]	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 [37]	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 [112]	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 [109]	SHA-1 hashing	N.A.
XenoSearch-II [108]	Hilbert space filling curve	N.A.
AdeepGrid [32]	SHA-1 hashing	N.A.
HP-protocol [5]	Reverse hilbert space filling curve	N.A.
Squid [103]	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 [20]	N.A.	$O((\log n)/k)$; k Long distance links; n is total peers, in a 1-dimensional search space
Adaptive [52]	Range search tree	$O(\log R_q)$; R_q is range selectivity, in a 1-dimensional search space
Kd-tree [51]	Kd-tree, skip pointer based on skip graphs	N.A.
Meghdoot [57]	SHA-1 hashing	N.A.
Z-curve [51]	Z-curves, skip pointer based on skip graphs	N.A.
P2PR-tree [80]	Distributed R-tree	N.A.
Super-P2P R*-Tree [72]	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 6: Classification based on No. of routing overlays for d-dimensional search space

Single	Multiple
JXTA search [54], DragonFly [29], XenoSearch-II [108], SWORD [83], Squid [103], Kd-tree [51],Meghdoot [57],Z-curve [51], QuadTree [113], P2PR-tree [80], Dgrid [114], AdeepGrid [32],Super-P2P R*-Tree [72]	PHT [88], MAAN [25], Adaptive [52], Pub/Sub-2 [116], P-tree [37],XenoSearch [109], Pub/Sub-1 [112], Mercury [20],HPPROTOCOL [5]

name, and constraints on possible values. Further, 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, then 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 (node at which publication and subscription request is 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 single attribute in a index digest (does not support multi-attribute range value in a single index digest).

5.1.2 XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform

XenoSearch [109] is a resource discovery system built for the XenoServer [58] execution platform. The XenoServer system is a Internet-based resource sharing platform that allows users to run programs at topologically distributed nodes. The XenoSearch indexes the resource information that are 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 (such as topological location, QoS attributes etc). The XenoSearch system is built over the Pastry [97] 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 co-ordinate 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) which summarizes the membership of ranges of nodes below them. These APs are identified by locations in the key space which 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 for 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 new search and data placement technique [108]. The new approach puts emphasis upon both the location and resource constraints associated with a search entity. Location constraints are

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

Algorithm Name	Heuristic Name	Preserves Data Locality (Yes/No)
PHT [88]	Chord routing	N.A.
MAAN [25]	Iterative resolution, single attribute dominated routing based on Chord	N.A.
Dgrid [114]	Chord routing	N.A.
SWORD [83]	Bamboo routing	No
JXTA search [54]	Broadcast .	N.A.
DragonFly [29]	Generic DHT routing	No
QuadTree [113]	Generic DHT routing	No
Pub/Sub-2 [116]	Chord routing	N.A.
P-tree [37]	Generic DHT routing	N.A.
Pub/Sub-1 [112]	Pastry routing	N.A.
XenoSearch [109]	Generic DHT routing	N.A.
XenoSearch-II [108]	Generic DHT routing	N.A.
AdeepGrid [32]	Single shot, recursive and parallel searching based on Pastry	No
HP-protocol [5]	Brute force, controlled flooding, directed controlled flooding based on CAN	N.A.
Squid [103]	Generic DHT routing	Yes
Mercury [20]	Range-selectivity based routing	N.A.
Adaptive [52]	Generic DHT routing	N.A.
Kd-tree [51]	Skip pointer based routing	Yes
Meghdoot [57]	CAN based routing	Yes
Z-curve [51]	Skip pointer based routing	Yes
P2PR-tree [80]	Block/group/subgroup pointer based routing	Yes
Super-P2P R*-Tree [72]	CAN based routing	Yes

defined using the primitives of disjunction ($\vee$), conjunction ($\wedge$), proximity ($near(A_1, A_2, \ldots, A_n)$), A_i denotes $i-$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 [100] data structure is used for the centralised 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 co-ordinate location system [107]. These d-dimensional co-ordinates are then mapped to a 1-dimensional linear index space using the Hilbert SFC.

5.1.3 AdeepGrid: Peer-to-Peer Discovery of Computational Resources for Grid Applications

AdeepGrid [32] 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-bit long as compared to 128-bit 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 that 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 by 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: (i) single-shot searching; (ii) recursive searching; and (iii) 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 then the node hosting the desired information replies with a REPLY (that contains resource information) message. On the other hand, recursive searching is a TTL (time to live) 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 that 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.

5.2 Chord Based Approaches

5.2.1 DGRID: A DHT-Based Grid Resource Indexing and Discovery Scheme

Work by Teo et al. [114] proposed a model for supporting GRIS over the Chord DHT. The unique characteristic about 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 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 [46], 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 i-bit identifier of t, where $i \leq m$, and $m - i$ 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 look-up operation in the DGRID is based on Chord look-up primitives. Given a key p, 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 look-up operation, a index S for the 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 identifier space is m-bit long. Overall, the look-up cost is bounded by the underlying Chord protocol i.e. $O(\log N)$. In general the look-up cost for a particular resource type t is $O(\log Y)$, Y is the total number of resource types available in the network.

5.2.2 Adaptive: An Adaptive Protocol for Efficient Support of Range Queries in DHT-based Systems

The work in [52] 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 the Chord for distributed routing and network management issues. A typical range query with length R_q is resolved by decomposing it into $O(\log(R_q))$ sub-queries. These sub-queries 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 a 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 replica of each other. Initially, a LBM has only one node but whenever the registration load on a particular node in the system exceeds a threshold (T_{reg}) then 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: $P = \lceil \frac{L^R}{C_R} \rceil$, 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 sub-ranges 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 sub-ranges. Typically, the range of the i-th node from the left represents the range $[v_i, v_{i+2^l-1}]$. 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 which 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 sub-queries, 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 which has the largest range within $[s, e]$ (also referred to as the node which 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 organisation techniques, more details on these aspects of the system can be found in the referenced article.

5.2.3 Pub/Sub-2: Content-based Publish-Subscribe Over Structured P2P Networks

The work in [116] 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 i-th dimension or a attribute a_i has a distinct data-type, name and value $v(a_i)$. A attribute type belongs to a predefined set of primitive data types commonly defined in most programming languages. A 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 which is 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 then 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. A 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_{a_i} 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 look-up search complexity.

5.2.4 QuadTree: Using a Distributed Quadtree Index in the Peer-to-Peer Networks

The work in [113] proposes a distributed quad-tree index that adopts an MX-CIF quadtree [100] 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 that are contained in or overlap 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 quad-tree representation. In general, the term quad-tree 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 upon the region quad-tree data structure. In this case, by applying the fundamental quad-tree 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 quad-tree index assigns regions of d-dimensional space to the peers in a P2P system. Every quad-tree block is uniquely identified by its centroid, termed as the control point. Using the control point, a quad-tree 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 quad-tree block, then it is responsible for processing all query computations that intersects the block. Multiple control points (i.e. quad-tree blocks) can be hashed to the same peer in the network. To avoid a single point of failure at the root level of the quad-tree the authors incorporate a technique called *fundamental minimum level, f_{min}*. This technique means that objects are only allowed to be stored at levels $l \geq f_{min}$ and 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 quad-tree 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 inturn call a subdivide() method that computes the intersecting control point associated with the new object or look-up 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 evokes 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 maintains a cache of addresses for its immediate children in the hierarchy. This reduces the subsequent look-up complexity to $O(1)$ beyond the root peer at 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 the average $O(\log_2 n)$ messages are required to locate a root peer for a query.

5.2.5 DragonFly: A Publish-Subscribe Scheme with Load Adaptability

The work in [29] 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 an d-dimensional Cartesian space. Every attribute in the system has lower and upper bound on its values. The bounds act as constraints for subscriptions and events 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 [100]. To negate a single point of failure at the root node, system adopts a technique called the *fundamental minimum level*. More details about this technique can be found in [113]. This technique recursively divides the logical space into four quadrants. With each recursion step on a existing quadrant, four new quadrants are generated. Hence, multiple recursion steps basically create a mutli-level quad tree data structure. The quad tree based organisation of DragonFly introduces parent-child relationships between tree cells. A cell at a 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 2-d space, the diagonal hyperplane is a line spanning from the north-west to the south-east vertices of the rectangular space. In d-dimensional context, this hyperplane is represented by the equation $\frac{x_1}{x_{max_1}-x_{min_1}} + \frac{x_2}{x_{max_2}-x_{min_2}} + \ldots + \frac{x_d}{x_{max_d}-x_{min_d}} = K$, where x_{max_d} and x_{min_d} are the upper and lower boundary values for d-th 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, then the child cells (peers at 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 untill all relevant child cells are notified for this subscription. However, if the root cell has not undergone any division process then 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 i.e. point and range event. 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 the subscription events is applied to locate the child cells.

5.2.6 MAAN: A Multi-Attribute Addressable Network for Grid Information Services

Cai et al. [25] present a multi-attribute addressable network (MAAN) approach for enabling a GRIS. They extend the Chord [111] protocol to support DRQs. MAAN addresses the d-dimensional range query problem by mapping the attribute values to the Chord identifier space via a uniform locality preserving hashing. Note that, for every attribute dimension a separate Chord overlay is maintained. For attributes with the 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 a attribute a, such that $l \leq v < u$, where l and u are the lower and upper bound respectively. In this case, node n formulates a look-up request and uses the underlying Chord routing algorithm to route it to node n_l such that $n_l = successor(H(l))$. The look-up 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 $successor(H(u))$ then it sends a reply message to the node n. Otherwise, the look-up request message is forwarded to its immediate successor until the request reaches the 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 $a_1, a_2, ..., a_m$ 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 $< attribute-value, resource-info >$ for different attributes. The resource look-up query in this case involves a multi-attribute range query which is a combination of sub-queries on each attribute dimension, i.e. $v_{il} \leq a_i \leq v_{iu}$ where $1 \leq i \leq M$, v_{il} and v_{iu} are the lower and upper bounds of the look-up query. MAAN supports two routing algorithms to resolve multiple-attribute queries: (i) iterative query resolution (IQR); and (ii) 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 look-up can be resolved in $O(\log N + N \times S_{min})$, where S_{min} is the minimum selectivity for all attributes.

5.2.7 Squid: Flexible Information Discovery in Decentralised Distributed Systems

Schmidt et al. [103] proposed a GRIS model that utilizes SFCs for mapping d-dimensional attribute space to a 1-dimensional search space. The proposed GRIS model consists of the following main components: (i) a locality preserving mapping that maps data elements to indices; (ii) an overlay network topology; (iii) a mapping from indices to nodes in the overlay network; (iv) a load balancing mechanism; and (v) 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 (*SFC*) [6], [63]. SFCs are used to generate a 1-dimensional 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 look-up operation involving partial queries and range queries typically requires interrogating more than one node, since the desired information is distributed across multiple nodes. The look-up queries can consist of combination of a 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 is needed to be looked up for range values. The query resolution process consists of two steps: (i) translating the attribute query to relevant clusters of the SFC-based index space and (ii) 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 a 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 virtual node is inherent to the Chord routing substrate.

5.2.8 P-tree: Querying Peer-to-Peer Networks Using P-trees

Crainniceanu et al. [37] 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 *left-most root-to-leaf path* of its corresponding $B^{+}-$ tree. The remaining part of the sub-tree 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 sub-tree and N is the total number of peers in the network. Overall, in a stable system when no inserts or deletes 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 1-dimensional 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 2-dimensional 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 properties. These properties includes 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 sub-trees at a level i have at least d non-overlapping entries at level $i - 1$. This ensures that the search cost is $O(\log_d N)$.

5.3 CAN Based Approaches

5.3.1 One torus to rule them all (Kd-tree and Z-curve based indexing)

The work in [51] proposes two approaches for enabling DRQs over the CAN DHT. The d-dimensional data is indexed using the well known spatial data structures: (i) z-curves ; and (ii) Kd-tree. First scheme is referred to as SCRAP: Space Filling Curves with Range Partitioning. SCRAP involves two fundamental steps: (i) the d-dimensional data is first mapped to a 1-dimensional using the z-curves; and (ii) 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: (i) mapping DRQ into SRQ using the SFCs; and (ii) 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 [7]. 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 a 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 2-d 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: (i) CAN DHT is used as basis for routing the DRQs ; (ii) 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 (iii) 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.

5.3.2 Meghdoot: Content-Based Publish/Subscribe over P2P Networks

The work in [57] 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 [99]. The model defines a d-dimensional attribute space given by the set $\mathcal{S} = A_1, A_2, \ldots, A_d$. Further, each attribute value A_i is denoted using the tuple Name:Type, Min, Max. Different Type includes a 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 =,$\geq$,$\leq$,$\geq$ and $\leq$) 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 $2d$-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. A 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 a 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.

5.3.3 HP-protocol: Scalable, Efficient Range Queries for Grid Information Services

Andrejak et al. [5] 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 $\frac{l+u}{2}$. 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 scheme 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.

5.3.4 Super-P2P R*-Tree: Supporting Multi-dimensional Queries in P2P Systems

The authors in the work [72] extend the d-dimensional index R*-tree [15], for supporting range and k-Nearest Neighbour (kNN) queries in a super-peer [122] based P2P system. The resulting distributed R*-tree is referred to as a NR-tree. 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 a index-cluster (i.e. a MBR), and can be controlled by one or more super-peer. Effectively, a 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 passive-peers, 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 sub-tree of entries with minimum min_dist.

5.4 Miscellaneous

5.4.1 SWORD: Distributed Resource Discovery on PlanetLab

SWORD [83] is a decentralised resource discovery service that supports multi-attribute queries. This system is currently deployed and tested over PlanetLab [33] 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 [94, 20] 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 [20].

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.

5.4.2 Mercury: Supporting Scalable Multi-Attribute Range Queries

Mercury [20] 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 $\mathcal{A}$. $\mathcal{A}_{\mathcal{Q}}$ and denotes the set of attributes in a query message using $\mathcal{Q}$. 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. A 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 $\mathcal{O} \log^2 n/k$ 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: i) successor and predecessor links within the local attribute hub; ii) k links to other nodes in the local attribute hub (intra-hub links) ; and iii) one link per hub (inter-hub link) that aids in communicating with other attribute hubs and resolving multi-attribute range queries. Note that, k intra-hubs 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 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 $[l_i, r_i]$. Key to message routing performance of Mercury is the choice of k intra-hub links. To set up each link i, a node draws a number $x \in \mathcal{I}$ 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 which 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 route to the value $\pi_a(D)$. For query $\mathcal{Q}$, $\pi_a(\mathcal{Q})$ is a range. In this case, first the message is routed to the first node that holds the starting range values and then range contiguity property is used to spread the query along the overlay network.

5.4.3 PHT: Prefix Hash Tree

The work in [88] presents a mechanism for implementing range queries over 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[1]. 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 [2]. The scheme associates a prefix label with each vertex of the tree. Given a vertex with label l, its left and right child vertices's are labeled as l_0 and l_1 respectively. The root of the tree is always labeled with the attribute name and all the subsequent vertexes are labeled recursively.

A data item is mapped and stored at the node having longest prefix match with the node label. A node can store upto B items, in case 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 look-up operation, a PHT node with label l is thus assigned to a node with identifier closest to HASH(l). Look-up for a range query in PHT network is performed by locating the node corresponding to the longest common prefix in the range. When such a node is found, then parallel traversal of its sub-tree is done to retrieve all the desired items. Note that significant query look-up speed-up can be achieved by dividing the range into a number of sub-ranges.

5.4.4 JXTA: JXTA Search

JXTA Search [119] is an open framework based on the JXTA [54] routing substrate. 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 point of contact for providers and consumers. Further each search hub is a member of a network of hubs which has at least one representative of hubs from every advertisement group. These groups are termed as *query groups*. Hence, in this case there is 100% 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 update message (new additions and deletions of registrations) to all the hub in its advertisement group. In case, 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 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, then the query is forwarded to that hub. In case the query cant be resolved in the local advertisement group then it is broadcasted to all remaining advertisement groups using a query group membership information. However, if the search network is organized based on content, then the query is routed to the advertisement group responsible for indexing the desired content.

5.4.5 P2PR-Tree: An R-Tree Based Spatial Index for P2P Environments

The work in [80] presents a scheme for adopting the R-tree [76] in a P2P setting. 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. Further, 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 on the subgroup) is done dynamically and are 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 which block(s) it belongs to.

When relevant block(s) are determined, a peer queries other peers in the same block for compiling group-related MBR information. It also queries atleast 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 following routing information: (i) pointers to all blocks in the universe; (ii) pointers to all groups in its block ; (iii) pointer to all subgroups in its group and ;(iv) finally pointers to all peers in its

[1] A trie is a multi-way 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.

[2] 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 which is a prefix of b

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

A query Q_L for a object is propagated recursively top down starting from level 0. When a query arrives at any peer P_i in the system, P_i checks whether its MBR covers the region indexed by the query. If so, then 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 untill the query block is located or the query reaches dead end of the tree.

6 Open Issues

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 a independent manner. They define standards without any proper coordination. They give very little attention to the inter-operatibility between the 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. In other words, a user from Condor-G grid can not submit his job to a Tycoon grid etc. 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 Internet component is the common resource indexing system i.e. 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 to inter-operate.

Possible solutions to overcome knowledge and resource fragmentation problem include: (i) availability of a robust, distributed, scalable resource indexing/organisation system; (ii) evolution of common standards for resource allocation and application superscheduling; (iii) agreement on using common middleware for managing grid resources such as clusters, SMPs etc; and (iv) defining common interfaces and APIs that can help different related system to inter-operate and coordinate activities. In recent times, we have seen some efforts towards developing a generic grid service-oriented architecture [61], more commonly referred to as Open Grid Service Architecture (OGSA). Core grid developers also define common standards through the GGF. In spite of all this, there is clearly a lack of global adoption of these standards, yet.

7 Summary and Conclusion

In the recent past, we have observed an increase in the complexity involved with grid resources including their management policies, organisation and scale. Key elements that differentiate a computational grid system from a PDCS include: (i) autonomy; (ii) decentralised ownership; (iii) heterogeneity in management policies, resource types and network inter-connect; and (iv) dynamicity in resource conditions and availability. Traditional grid systems [48, 8, 2] based on centralised information services are proving to be bottleneck with regard to scalability, fault-tolerance and mechanism design issues. To address this, P2P based resource organisation is being advocated. P2P organisation is scalable, adaptable to dynamic network conditions and highly available.

In this work, we presented a detailed taxonomy that characterizes issues involved in designing a P2P/decentralised GRIS. We classified the taxonomies into two sections: (i) resource taxonomy; and (ii) P2P taxonomy. Our resource taxonomy highlighted the attributes related to a computational grid resource. Further, we summarized 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 presented classification of P2P approaches based on three dimensions including: (i) P2P network organisation; (ii) approaches to distribution of the data among the peers; and (iii) routing of d-dimensional queries. In principle, data distribution mechanism directly dictates how a query is routed among the relevant peers. D-dimensional resource index is distributed among peers by utilizing the data structures such as SFCs, quad-trees, R-trees and Kd-trees. Some of the approaches have also modified existing hashing schemes to facilitate the 1-dimensional range queries in a DHT network. Every approach has its own merits and limitations. Some of these issues were highlighted in the resource and P2P network organisation taxonomy section.

However, a few questions that still remain to be answered include: (i) which of the surveyed approach is best suited for organizing a GRIS? (ii) do current approaches provide enough flexibility in query composition and resolution that is required to undertake complexity of Grid superscheduling systems? (iii) is there a need to define a new query composition language which is capable of representing all possible kinds of queries present now or that could arise in future? and (iv) are the current DHT based P2P techniques are robust enough to support such complex query systems?

8 ACKNOWLEDGMENTS

Firstly, we should like to acknowledge the authors of the papers whose work have been surveyed and utilized in developing the taxonomy in this paper. 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 Lab, at the University of Cambridge), David Spence (CCLRC Rutherford Appleton Laboratory, e-Science Center, Oxford, UK), and Manish Parashar (Professor of Electrical and Computer Engineering, Rutgers University) for their valuable feedbacks on the initial version of the paper. Special thanks to Krishna Nandiminti and Hussein Gibbins for proof-reading the paper. Both Krishna and Hussein work as Research Programmer in the GRIDS Lab, University of Melbourne. We are also grateful to the DEST (Department of Education, Science and Training, Australian Government) and University of Melbourne for funding the PHD candidature of Rajiv Ranjan. This work is also supported through Australian Research Council discovery project grant.

References

[1] S. Surana R. Karp A. Rao, K. Lakshminarayanan and I. Stoica. Load balancing in structured p2p systems. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[2] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems (FGCS) Journal, Volume 18, Issue 8, Pages: 1061-1074, Elsevier Science, The Netherlands, October*, 2002.

[3] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. OurGrid: An approach to easily assemble grids with equitable resource sharing. In *Proceedings of the 9th Workshop on Job Scheduling Strategies*. Lecture Notes in Computer Science, 2003.

[4] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. Tortone, and C. Vistoli. GridICE: a monitoring service for the grid. *Proceedings of the Third Cracom Grid Workshop*, pages 220–226, 2003.

[5] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *P2P'02: Second IEEE International Conference on Peer-to-Peer Computing*, Linkkoping, Sweden, 2002. IEEE.

[6] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer. Space-filling curves and their use in the design of geometric data structures. *Theor. Comput. Sci.*, 181(1):3–15, 1997.

[7] J. Aspnes and G. Shah. Skip graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384–393, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[8] A. Auyoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *OASIS '04: 1st Workshop on Operating System and Architectural Support for the Ondemand IT InfraStructure, Boston, MA, October*, 2004.

[9] M. Baker and G. Smith. GridRM: a resource monitoring architecture. In *Grid Computing - GRID 2002 : Third International Workshop, Baltimore, MD, USA*, pages 268–273. LNCS, 2002.

[10] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: A survey, citeseer.ist.psu.edu/bakhtiari95cryptographic.html, 1995.

[11] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in p2p systems. *Commun. ACM*, 46(2):43–48, 2003.

[12] Z. Balaton, P. Kacsuk, N. Podhorszky, and F. Vajda. From cluster monitoring to grid monitoring based on grm. In *Euro-Par'01-7th international Euro-Par conference*, pages 874–881, Washington, DC, USA, 2001. IEEE Computer Society.

[13] M. Balazinska, H. Balakrishnan, and D. Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery, 2002.

[14] D. Barbara. Mobile computing and databases-a survey. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):108–117, 1999.

[15] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 322–331, New York, NY, USA, 1990. ACM Press.

[16] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[17] S. Berchtold, C. Bohm, and Hans-Peter Kriegal. The pyramid-technique: towards breaking the curse of dimensionality. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 142–153, New York, NY, USA, 1998. ACM Press.

[18] M. J. Berger and S. H. Bokhari. A partitioning strategy for non-uniform problems on multiprocessors. In *IEEE Trans. Computers*, pages 570–580, 1987.

[19] K. A. Berman and J. L. Paul. *Fundamentals of Sequential and Parallel Algorithms*. PWS Publishing Company, 1997.

[20] A. Bharambe, M. Agarwal, and S. Seshan. MERCURY: supporting scalable multi-attribute range queries. In *SIGCOMM'04: In Proceedings of SIGCOMM 2004*. ACM, 2004.

[21] M. Bienkowski, M. Korzeniowski, and F. M. auf der Heide. Dynamic load balancing in distributed hash tables. In *IPTPS*, pages 217–225, 2005.

[22] B. Bode, D. Halstead, R. Kendall, and D. Jackson. PBS: The portable batch scheduler and the maui scheduler on linux clusters. *Proceedings of the 4th Linux Showcase and Conference, Atlanta, GA, USENIX Press, Berkley, CA, October*, 2000.

[23] F. Bonnassieux, R. Harakaly, and P. Primet. MapCenter: an open grid status visualization tool. *Proceedings of the ICSA 15th International Conference on Parallel and Distributed Computing Systems*, 2002.

[24] A. Raza Butt, R. Zhang, and Y. C. Hu. A self-organizng flock of condors. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2003. IEEE Computer Society.

[25] M. Cai, M. Frank, J. Chen, and P. Szekely. Maan: A Multi-atribute addressable network for grid information services. *Proceedings of the Fourth IEEE/ACM International workshop on Grid Computing;Page(s):184 - 191*, 2003.

[26] K. Calvert, M. Doar, and E. W. Zegura. Modeling internet topology. In *IEEE Communications Magazine*, Washington, DC, USA, June 1997. IEEE Computer Society.

[27] M. Castro, M. Costa, and A. Rowstron. Should we build gnutella on a structured overlay? *SIGCOMM Comput. Commun. Rev.*, 34(1):131–136, 2004.

[28] M. Castro, P. Druschel, A.M. Kermarrec, and A. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[29] L. Chan and S. Karunasekera. Dragonfly: A publish-subscribe scheme with load adaptability. Technical report, Department of Computer Science and Software Engineering, The University of Melbourne, 2006.

[30] S. Chapin, J. Karpovich, and A. Grimshaw. The legion resource management system. *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, San Juan, Puerto Rico, 16 April, Springer:Berlin*, 1999.

[31] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, New York, NY, USA, 2003. ACM Press.

[32] A. S. Cheema, M. Muhammad, and I. Gupta. Peer-to-peer discovery of computational resources for grid applications. In *Grid'05: 6th IEEE/ACM International Workshop on Grid Computing*, Washington, DC, USA, 2005. IEEE.

[33] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[34] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International workshop on Designing privacy enhancing technologies*, pages 46–66, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[35] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[36] C. Courcoubetis and V. Siris. Managing and pricing service level agreements for differentiated services. In *Proc. of 6th IEEE/IFIP International Conference of Quality of Service (IWQoS'99), London, UK, May-June*, 1999.

[37] A. Crainniceanu, P. Linga, J. Gehrke, and J. Shanmugasundram. Querying peer-to-peer networks using p-trees. In *'04: Seventh International Workshop on the Web and Databases*. ???, 2004.

[38] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, page 181, Washington, DC, USA, 2001. IEEE Computer Society.

[39] K. Czajkowski, I. Foster, and C. Kesselman. Agreement-based resource management. In *Proceedings of the IEEE, Vol.93, Iss.3, March 2005*, Washington, DC, USA, 2005. IEEE Computer Society.

[40] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA, 2001. ACM Press.

[41] P. A. Dinda, T. Gross, R. Karrer, B. Lowekamp, N. Miller, P. Steenkiste, and D. Sutherland. The architecture of the remos system. *hpdc*, 00:0252, 2001.

[42] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. *hotos*, 00:0075, 2001.

[43] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, and H. Truong. Askalon: a tool set for cluster and grid computing: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):143–169, 2005.

[44] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, pages 365–375. IEEE Computer Society Press, 1997.

[45] G. H. Forman and J. Zahorjan. The challenges of mobile computing. *Computer*, 27(4):38–47, 1994.

[46] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications, 11(2), Pages:115-128*, 1997.

[47] I. Foster and C. Kesselman. The grid: Blueprint for a new computing infrastructure. *Morgan Kaufmann Publishers, USA*, 1998.

[48] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01), 2001*, pages 237 – 246, Washington, DC, USA, 2001. IEEE Computer Society.

[49] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[50] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. Technical report, Stanford U., 2004.

[51] P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: multi-dimensional queries in p2p systems. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 19–24, New York, NY, USA, 2004. ACM Press.

[52] J. Gao and P. Steenkiste. An adaptive protocol for efficient support of range queries in dht-based systems. In *ICNP '04: Proceedings of the Network Protocols, 12th IEEE International Conference on (ICNP'04)*, pages 239–250, Washington, DC, USA, 2004. IEEE Computer Society.

[53] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. *ccgrid*, 00:35, 2001.

[54] L. Gong. JXTA: a network programming environment. *IEEE Internet Computing*, 05(3):88–95, 2001.

[55] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. Netlogger: A toolkit for distributed system performance analysis. *mascots*, 00:267, 2000.

[56] A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, pages 7–12, Lihue, Hawaii, May 2003.

[57] A. Gupta, O. D. Sahin, D. Agrawal, and A. El. Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 254–273, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[58] S. Hand, T. Harrisand E. Kotsovinos, and I. Pratt. Controlling the xenoserver open platform. In *2003 IEEE Conference on Open Architectures and Network Programming,*, pages 3–11, Washington, DC, USA, 2003. IEEE Computer Society.

[59] R. Huebsch, J. M. Hellerstein, N. L. Boon, T. L., S. Shenker, and I. Stoica. Querying the internet with pier.

[60] P. Linga A. Demers I. Gupta, K. Birman and R. van. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, 2003.

[61] A. Iamnitchi and I. Foster. On fully decentralized resource discovery in grid environments. *International Workshop on Grid Computing, Denver, CO*, 2001.

[62] A. Iamnitchi and I. Foster. A peer-to-peer approach to resource location in grid environments. pages 413–429, 2004.

[63] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 332–342, New York, NY, USA, 1990. ACM Press.

[64] H. V. Jagadish. Analysis of the hilbert curve for representing two-dimensional space. *Inf. Process. Lett.*, 62(1):17–22, 1997.

[65] M.A. Jovanovic, F.S. Annexstein, and K.A. Berman. Scalability issues in large peer-to-peer networks-a case study of gnutella. Technical report, University of Cincinnati, 2001.

[66] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663, New York, NY, USA, 1997. ACM Press.

[67] F. Korn, B. Pagel, and C. Faloutsos. On the 'dimensionality curse' and the 'self-similarity blessing'. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):96–111, 2001.

[68] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

[69] K. Lai, B. A. Huberman, and L. Fine. Tycoon: A distributed market-based resource allocation system. *Technical Report, HP Labs*, 2004.

[70] J. Li, J. Stribling, T. M. Gil, R. Morris, and M. Frans Kaashoek. Comparing the performance of distributed hash tables under churn. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS04)*, San Diego, CA, February 2004.

[71] J. Litzkow, M. Livny, and M. W. Mukta. Condor- a hunter of idle workstations. *IEEE*, 1988.

[72] B. Liu, W. Lee, and D. L. Lee. Supporting complex multi-dimensional queries in p2p systems. In *ICDCS'05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 155– 164, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[73] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. In *Communications Surveys and Tutorials*, Washington, DC, USA, 2005. IEEE.

[74] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal. Peer-to-peer grid computing and a .net-based alchemi framework, high performance computing: Paradigm and infrastructure. 2004.

[75] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM Press.

[76] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Yannis Theodoridis. R-trees have grown everywhere.

[77] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, 1995.

[78] D.S. Milojicic, V. Kalogeraki, R. Lukose, and K. Nagarajan. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, 2002.

[79] A. Mislove and A. Post. Post: A secure, resilient, cooperative messaging system, 2003.

[80] A. Mondal, Y. Lifu, and M. Kitsuregawa. P2PR-Tree: An r-tree-based spatial index for peer-to-peer environments. In *EDBT 2004*, pages 516 – 525, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[81] H.B. Newman, I.C. Legrand, P.Galvez, R. Voicu, and C. Cirstoiu. MonALISA: a distributed monitoring service architecture. *Proceedings of the CHEP 2003*, 2003.

[82] N. F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70, 2004.

[83] D. Oppenheimer, J. Albrecht, A. Vahdat, and D. Patterson. Design and implementation tradeoffs for wide-area resource discovery. In *Proceedings of 14th IEEE Symposium on High Performance, Research Triangle Park, NC*, Washington, DC, USA, July 2005. IEEE Computer Society.

[84] J. Orenstein. A comparison of spatial query processing techniques for native and parameter spaces. In *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 343–352, New York, NY, USA, 1990. ACM Press.

[85] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In *Proceedings of the European Grid Conference*. Lecture Notes in Computer Science, Springer-Verlag, 2005.

[86] G. Pallis and A. Vakali. Insight and perspectives for content delivery networks. *Commun. ACM*, 49(1):101–106, 2006.

[87] B. Preneel. The state of cryptographic hash functions. In *Lectures on Data Security, Modern Cryptology in Theory and Practice, Summer School, Aarhus, Denmark, July 1998*, pages 158–182, London, UK, 1999. Springer-Verlag.

[88] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief announcement: Prefix Hash Tree. In *In Proceedings of ACM PODC*, New York, NY, USA, 2004. ACM Press.

[89] R. Raman, M. Livny, and M. Solomon. Matchmaking: distributed resource management for high throughput computing. *High Performance Distributed Computing, 28-31 July*, 1998.

[90] R. Ranjan, R. Buyya, and A. Harwood. A case for cooperative and incentive based coupling of distributed clusters. In *Proceedings of the 7th IEEE International Conference on Cluster Computing (CLUSTER'05), Boston, MA*.

[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] R. Ranjan, A. Harwood, and R. Buyya. A SLA-based coordinated superscheduling scheme and performance for computational grids. *Technical Report, GRID S-TR-2006-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia*, 2006.

[93] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[94] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: a public DHT service and its uses. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–84, New York, NY, USA, 2005. ACM Press.

[95] R. L. Ribler, J. S. Vetter, Huseyin Simitci, and Daniel A. Reed. Autopilot: Adaptive control of distributed applications. In *HPDC'98: Proceedings of the Seventh International Symposium on High Performance Distributed Computing (HPDC-7)*, pages 172–179, 1998.

[96] R.L. Rivest. Partial match retrieval algorithms. *SIAM Journal of Computing*, 5(1):19–50, 1976.

[97] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware'01: Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–359, Heidelberg, Germany, 2001.

[98] F.D. Sacerdoti, M.J. Katz, M.L. Massie, and D.E. Culler. Wide area cluster monitoring with ganglia. In *Proceedings of the 5th IEEE International Conference on Cluster Computing (CLUSTER'03), Tsim Sha Tsui, Kowloon, Hong Kong*.

[99] O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. A peer-to-peer framework for caching range queries. *ICDE*, 00:165, 2004.

[100] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Publishing Company, 1989.

[101] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems, 2002.

[102] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[103] C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. *In the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), June*, 2003.

[104] J.M. Schopf. Ten actions when superscheduling. In *Global Grid Forum*, 2001.

[105] H. Shan, L. Oliker, and R. Biswas. Job superscheduler architecture and performance in computational grid environments. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 44, Washington, DC, USA, 2003. IEEE Computer Society.

[106] W. Smith. A system for monitoring and management of computational grids. In *Proceedings of the 31st International Conference on Parallel Processing (ICPP2002)*, Washington, DC, USA, 2002. IEEE.

[107] D. Spence. An implementation of a coordinate based location system. Technical report, University of Cambridge, Computer Laboratory, 2003.

[108] D. Spence, J. Crowcroft, S. Hand, and T. Harris. Location based placement of whole distributed systems. In *CoNEXT'05: Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, pages 124–134, New York, NY, USA, 2005. ACM Press.

[109] D. Spence and T. Harris. Xenosearch: Distributed resource discovery in the xenoserver open platform. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, page 216, Washington, DC, USA, 2003. IEEE Computer Society.

[110] P. Stelling, C. Lee, I. Foster, G. von Laszewski, and C. Kesselman. A fault detection service for wide area distributed computations. In *HPDC '98: Proceedings of the The Seventh IEEE International Symposium on High Performance Distributed Computing*, page 268, Washington, DC, USA, 1998. IEEE Computer Society.

[111] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. Frans Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *To Appear in IEEE/ACM Transactions on Networking*, 2002.

[112] D Tam, R Azimi, and HA Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. In *International Workshop on Databases, Information Systems and Peer-to-Peer Computing*. springerlink, 2003.

[113] E. Tanin, A. Harwood, and H. Samet. A distributed quadtree index for peer-to-peer settings,. In *In Proceedings of the International Conference on Data Engineering - ICDE*, pages 254–255, 2005.

[114] Y.M. Teo, V. March, and X. Wang. A DHT-based grid resource indexing and discovery scheme. In *Singapore-MIT Alliance Annual Symposium*, 2005.

[115] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture. *Global Grid Forum*, 2002.

[116] P. Triantafillou and I. Aekaterinidis. Content-based publish/-subscribe over structured p2p networks. *1st International Workshop on Discrete Event-Based Systems*, 2004.

[117] H. L. Truong and T. Fahringer. SCALEA-G: a unified monitoring and performance analysis system for the grid. *LNCS*, pages 202–211, 2004.

[118] 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*, 18(6):685–699, 2006.

[119] S. Waterhouse, D. M. Doolin, G. K., and Y. Faybishenko. Distributed search in p2p networks. *IEEE Internet Computing*, 06(1):68–72, 2002.

[120] R. Wismuller, J. Trinitis, and T. Ludwig. OCM-a monitoring system for interoperable tools. In *SPDT '98: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*, pages 1–9, New York, NY, USA, 1998. ACM Press.

[121] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: the network weather service. In *Supercomputing '97: Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–19, New York, NY, USA, 1997. ACM Press.

[122] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *ICDE'03: Proceedings of the 19th IEEE International Conference on Data Engineering*, volume 00, page 49, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[123] C. S. Yeo, R. Buyya, M. D. Assuncao, J. Yu, A. Sulistio, S. Venugopal, and M. Placek. Utility Computing on Global Grids. In Hossein Bidgoli, editor, *Handbook of Computer Networks*. John Wiley and Sons.

[124] J. Yu and R. Buyya. A novel architecture for realizing grid workflow using tuple spaces. In *Grid'04: Proceedings of 5th IEEE/ACM Grid Workshop*, pages 119–128, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[125] J. Yu, S. Venugopal, and R. Buyya. Grid market directory: A web and web services based grid service publication directory. *The Journal of Supercomputing*, 36(1):17–31, 2006.

[126] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems (FGCS) Journal, Volume 21, Issue 1, Pages: 163-188, Elsevier Science, The Netherlands, January*, 2005.

[127] X. Zhang, J. L. Freschl, and J. M. Schopf. A performance study of monitoring and information services for distributed systems. *In the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), June*, 2003.

[128] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[129] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. *In Proceedings of the Workshop on Cluster Computing, Tallahassee, FL,*, 1992.