Decentralized Resource Discovery Mechanisms for Distributed Computing in Peer-to-Peer Environments

DANIEL LAZARO, Open University of Catalonia
JOAN MANUEL MARQUES, Open University of Catalonia and Technical University of Catalonia
JOSEP JORBA and XAVIER VILAJOSANA, Open University of Catalonia

Resource discovery is an important part of distributed computing and resource sharing systems, like grids and utility computing. Because of the increasing importance of decentralized and peer-to-peer environments, characterized by high dynamism and churn, a number of resource discovery mechanisms, mainly based on peer-to-peer techniques, have been presented recently. We present and classify them according to criteria like their topology and the degree of achievement of various common requirements of great importance for the targeted environments, as well as compare their reported performance. These classifications intend to provide an intuitive vision of the strengths and weaknesses of each system.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and Survey; C.2.4 [**Computer-communication Networks**]: Distributed Systems

General Terms: Algorithms, Design, Performance, Reliability, Security

Additional Key Words and Phrases: Ad hoc grids, overlay networks, P2P networks, resource discovery, utility computing

ACM Reference Format:
Lazaro, D., Marques, J. M., Jorba, J., and Vilajosana, X. 2013. Decentralized resource discovery mechanisms for distributed computing in peer-to-peer environments. ACM Comput. Surv. 45, 4, Article 54 (August 2013), 40 pages.
DOI: http://dx.doi.org/10.1145/2501654.2501668

1. INTRODUCTION

Distributed computing and resource sharing systems have to deal with a number of concerns, including group membership, security, and others. One that is of high importance is resource discovery. It allows participants (users, agents, or other components of the system) to find resources in the network that satisfy certain requirements. In the most general case, resource discovery can include finding any type of resource, from searching files shared by members of a group to services located in specific computers, among others. In this article, we focus on a specific category of resource discovery mechanisms: those for finding computational resources.

This work has been partially supported by the "Comissionat per a Universitats i Recerca del Departament d' Innovacio, Universitats i Empresa de la Generalitat de Catalunya" and by the "Fons Social Europeu" under grant FI, by TIN2008-01288/TSI and by TIN2007-68050-C03-01.
Authors' addresses: D. Lazaro (corresponding author), Open University of Catalonia, Avinguda del Tibidabo, 39 08035 Barcelona, Spain; email: dlazaro@uoc.edu; J. M. Marques, Open University of Catalonia and Technical University of Catalonia, Calle Jordi Girona, 31 08034 Barcelona, Spain; J. Jorba and X. Vilajosana, Open University of Catalonia, Avinguda del Tibidabo, 39 08035 Barcelona, Spain.

DOI: http://dx.doi.org/10.1145/2501654.2501668

We refer to resources as sets of computational resources that can be used to execute a piece of code, whether it is a short computational job or a permanent service. In short, a resource would be either a physical machine or a virtual machine. Other uses for resources, like storage or provision of specific services, might also be supported. An important factor is that they are in a fixed location, so the system cannot autonomously distribute and replicate them as would be done with files and other examples of more general definitions of resources. We call them indistinctly resources, machines, or nodes throughout this article.

By resource discovery we specifically mean finding a set of resources whose capacities and characteristics are described in a specification published by their owners that satisfy a given set of requirements. Once these resources have been found, they can be used for various purposes, like executing a job or installing a service.

Resource discovery is a very important part of grid systems. These systems aim at sharing resources among different physical organizations, forming Virtual Organizations or VOs that can use each other's resources to perform computational tasks following a pre-agreed set of policies. Because of their potentially large size, grids require a resource discovery system. However, grids usually have enough stable resources to implement centralized resource discovery mechanisms. Such solutions can be very efficient, provided that the required infrastructure is available.

Besides grids, there are also other kinds of distributed computing systems with more open approaches, like peer-to-peer (p2p) systems, ad hoc grids, utility computing systems, etc. In these systems, provisioning of dedicated or well-known servers might be difficult or even inconvenient, depending on the characteristics of the communities and the internal policies that define them. In many cases, the same resources that are available for execution of tasks must also manage the discovery mechanisms without the support of dedicated servers that would allow a centralized approach. In other situations, the size of the system is such that a great number of dedicated servers are needed to efficiently perform resource discovery. Therefore, in all these cases a great number of nodes must be coordinated to provide the capacity of finding resources in a large-scale network.

When designing a resource discovery system, it is important to determine the kind of environment in which it is intended to work, as well as the type and quality of service that will be acceptable from it. This will determine the specific requirements that it must accomplish, both functional and nonfunctional. Some systems will require highly scalable mechanisms. Some will require low response times, while others will put an emphasis on completeness and accuracy of returned results, or in minimizing network traffic overhead.

Depending on its specific requirements, there are a number of design decisions that can achieve their fulfillment and build an adequate resource discovery mechanism for the targeted environment. In this survey we present a number of systems aimed at similar environments, that is, decentralized dynamic systems for remote execution of jobs and/or services. We identify the main requirements that these systems can have, and propose a classification of the different design decisions according to some common criteria. We analyze the requirements fulfilled by each of these systems and classify them. The comparison of the different systems according to our parameters is presented in tables and graphs for easier visibility. With all these, we expect to provide a guidance on the relative pros and cons of each of them, and help researchers and practitioners to choose the right system for their targeted environment.

Some of the requirements and design characteristics considered in our survey are also considered as p2p discovery properties by Surarayana and Taylor [2004], although their analysis is focused on p2p document discovery, while ours is focused on discovery of computational resources. Ranjan et al. [2008] also present a number of taxonomies

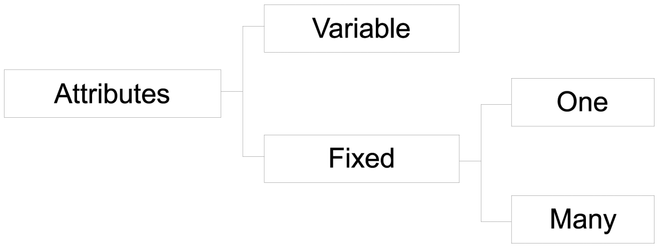

Fig. 1. Resource description taxonomy.

classifying different aspects of p2p resource discovery, some of which coincide partially with our own work. However, unlike them, we do not focus only on classifying systems through taxonomies, but also on analyzing the fulfillment of specific requirements by each system and how this is related to the design characteristics that define them. Therefore, our classifications do not exactly adhere to either of these previously proposed models, but try instead to take the most relevant parts of them and augment them where needed for our purpose.

The remainder of this article is structured as follows. Section 2 presents the main requirements that are addressed by decentralized resource discovery systems. Section 3 presents the main design characteristics that define a resource discovery mechanism. Section 4 presents descriptions of a number of decentralized resource discovery mechanisms and systems which have been proposed by the research community. Section 5 compares the presented systems according to the requirements they fulfill and their design. Section 6 draws some final conclusions and ends the article.

2. REQUIREMENTS

In this section we present the main requirements that are addressed, in different degrees, by decentralized resource discovery systems.

2.1. Search Flexibility

In a resource discovery service, users need to express what resources they need in a way that is convenient for them and for the activities they want to perform with the available resources. For example, in a distributed storage system, the requirements for resources could include only physical storage capacity and available network connection. However, the system could also require proximity between the user and the storage node. In a job execution system, users may need to specify other characteristics like CPU capacity. Moreover, more complex tasks composed of several distributed, interconnected processes might require a set of nodes with specific inter-node conditions, like proximity.

By search flexibility, we refer to the ability of a resource discovery system to search resources satisfying different kinds of criteria. Some examples are searching for specific values on multiple attributes, values into a given range for one or more attributes, or checking inter-node relations. However, search flexibility requires both being able to handle complex queries and simple ones.

The flexibility is defined by two factors: how the resources are described, and what kind of searches can be performed over the resources. We present a taxonomy for each of these factors. These are similar to the ones presented by Ranjan et al. [2008].

Resources are described by a series of attributes and their values. These attributes can be of different types: numerical, boolean, textual, etc. Each system allows different degrees of flexibility at the time of describing resources. This is presented as a taxonomy in Figure 1. The most flexible systems allow for a variable amount of attributes to

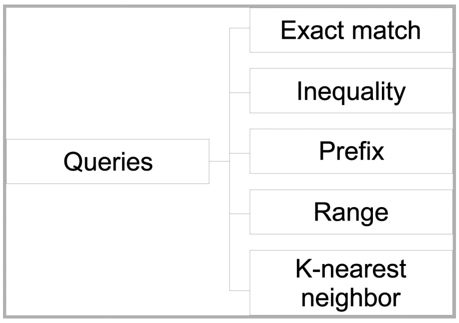

Fig. 2. Resource discovery query type taxonomy.

define each resource, while others only can deal with a fixed schema. The former can allow resources to freely publish the presence of attributes other resources may not have, like rare devices, as well as allow them to provide only a partial specification, omitting the values for some attributes. The latter, on the other hand, requires a fixed set of attributes, all of which must have values specified by resources. This is usually necessary to index the information for its better processing. Some systems are only able to deal with one attribute, while others can process any number of attributes, provided that it is fixed statically.

This is the most relevant division according to the resource description, since different types of attributes can be converted to other types, usually numerical, to be treated by the resource discovery system. However, different types might be more adequate to certain search mechanisms, or their presence might skew value distributions. Therefore, it is recommendable to be careful with this factor.

Figure 2 shows a taxonomy of different types of queries supported by resource discovery systems. The exact match queries are the most limited, since they can only return resources that match exactly the value introduced by the user. This is usually of little use in resource discovery for computational purposes, since applications often have a set of minimum requirements that can be exceeded without inconvenience. However, this approach is far easier to implement than the others. Inequality operators can be useful for the aforementioned case where an application specifies a minimum set of requirements, with no explicit maximum limit, and can be easier to implement than range queries. Prefix queries allow to look for values, expressed in a string (usually of bits) form, that have a given prefix. This is a case where the presence of different attribute types can make this approach more or less adequate than range queries, where resources with attribute values inside a given range are returned. Other types of queries can be processed by the system, like k-nearest neighbor (finding the k nodes whose value is nearer to a given one).

Finally, our taxonomies do not include the possibility of inter-node requirements. Despite their undeniable importance, only one of the reviewed systems includes these, and they are evaluated after receiving the data from the distributed index by processing the possible combinations among the returned resources. Therefore, inter-node requirements do not appear to have an influence on the design and implementation of distributed indexes for resource discovery systems. This is a point where decentralized resource discovery mechanisms are behind centralized ones, since many of these offer the possibility of inter-node requirements. One such centralized system is Condor [Thain et al. 2005]. This is naturally easier in centralized systems since all information about resources is available at one place to analyze and find a set fulfilling inter-node requirements. Performing this kind of analysis and selection in a decentralized manner remains an open problem.

2.2. Scalability

Scalable mechanisms and architectures adapt to groups of large sizes, with a high number of resources and/or clients, while still keeping a good quality of service, in whatever terms it is defined. This affects aspects like response time or overhead regarding storage, network, or computational load.

The required degree of scalability depends on the intended environment of the system and the conditions under which it is expected to work. A very high scalability can allow a system to be deployed in a wide variety of systems, but some mechanisms can trade scalability for better performance in small or medium size systems, so the target environment is decisive.

Very related to scalability is the term, frequently used in overlay networks, of lookup complexity. It denotes the quantity of messages, with respect to number of nodes in the system or other size metrics, necessary to perform a search.

2.3. Churn and Fault Tolerance

In peer-to-peer systems, ad hoc grids, and similar systems, it is usual to have a relatively important amount of churn, that is, connections and disconnections of nodes in the system. This can be due to users turning off their machines when they are not working, taking them out of the system for personal use, or failures that cause the machine to not be available to the system. Therefore, it is desirable for a resource discovery system aimed at this kind of environments to be able to deal with churn and failures. This means that the system will recover a correct state, where it is able to discover resources in the ordinary conditions of completeness and performance, within a short time and without a great overhead. Byzantine failures are considered inside the category of security.

2.4. Completeness

This refers to the ability of a resource discovery mechanism to find all resources available in its system that satisfy a certain set of requirements. However, real applications might not need to discover all resources that meet their requirements, but only a set of machines that can allow them to carry out their activities. On the other hand, complete results can be used to select the best assignment of resources for each task in order to achieve an optimal global usage of the system.

A mechanism that is not complete can fail to discover a resource that is present in the system. Therefore, it might be unable to answer a query for resources issued by a user, even in the case where the user does not require complete results. Nevertheless, in an environment where resources meeting the usually required properties are expected to be common, an incomplete mechanism can often offer a high probability of finding resources while achieving better performance is better than a complete one.

2.5. Accuracy

Not only is it important whether a mechanism can find all the available resources meeting a set of requirements or not, but also whether the results returned actually meet all the specified requirements or not. This is what we call accuracy of results. If accuracy is not total, a client will need to perform additional verification of returned results in order to be sure of their adequacy to the intended use. However, when the complexity of requirements is high, it might be a more adequate solution to provide a lower accuracy in results. Obtaining a list of resources that might meet the requirements and processing it to find those which actually can yield a better performance is better than checking many complex conditions in a distributed mechanism.

2.6. Security

In distributed systems, and especially in those that are open, security is an extremely important concern. There are many aspects of a system where security concerns arise, including user identification and membership, access rights and resistance to denial-of-service attacks, among many others.

Security is not a primary concern in the resource discovery systems that are represented in this survey. However, tolerance to attacks and to the presence of malicious nodes is important for discovering resources in an open environment. For example, nodes can provide false or inaccurate descriptions of their capacities, allowing them to be selected for some tasks that they will not be able to perform. Moreover, many mechanisms give some nodes control over what resources are selected for answering a query. A malicious node in that position can force a user to employ malicious nodes for his or her tasks, giving them control over the results and data related to that task. Even if there are other mechanisms that allow the user to identify these malicious nodes, like trust or reputation systems, one might still be able to deny discovery of legitimate resources.

The security of a system must be evaluated as a whole, considering the vulnerability of all its components and the targeted environment. For example, a strict policy for member admission can cause relaxed policies in other components of the system to be sufficient. However, because of the existence of cases like the one exposed earlier, we believe that resource discovery mechanisms can be the target of specific attacks, especially in open environments like ad hoc grids and peer-to-peer systems. Therefore, resistance to Byzantine failures could be a requirement for resource discovery systems.

2.7. Miscellaneous Performance Requirements

Different metrics can be considered to evaluate the performance of a system, and different systems may give a different importance to each metric. For example, it is a common objective to achieve a lower cost for more frequent operations. A system where queries are constant and the characteristics of the resources do not change frequently will gladly sacrifice a higher cost in updating resource information in exchange for optimizing the performance of queries. Meanwhile, for a system where the capacity of the resources changes rapidly, it will be preferable to achieve an acceptable level of performance for both queries and updates, rather than sacrificing one for the other.

There are other common performance trade-offs in the design of resource discovery systems. Some systems may give great importance to query latency, while others may give a higher priority to the storage or communication cost of the mechanisms (i.e., information stored by nodes, periodic messages, etc.). All these requirements will affect the decisions taken in different parts of the design.

3. DESIGN CHARACTERISTICS

We present some characteristics that must be considered when designing a resource discovery mechanism. Decisions taken about each characteristic may influence the achievement of a specific set of requirements, although the combination of all the design decisions will ultimately determine the degree of fulfillment of the requirements.

3.1. Overlay Topology

An overlay network [En Keong Lua et al. 2005] is a set of nodes interconnected by logical connections that work over a physical network. Each of these logical links can abstract an arbitrary number of physical links. This concept is broadly used in peer-to-peer and decentralized systems, including resource discovery systems. Overlay networks can be classified according to their architecture into various categories, which determine many of their characteristics.

Some systems form their overlay network by using what is usually called supernodes or superpeers. That means that a subset of the nodes are connected among them in a decentralized overlay network, while the rest only have a connection with one of the supernodes, usually the one nearer them according to some metric (geographical proximity, semantical similarities, etc.), in a centralized server-client fashion. In these cases, we refer as overlay network to the one formed by the so-called supernodes that form a decentralized network and, in a sense, represent the rest of the nodes.

3.1.1. Structured. These topologies present a certain predetermined structure which guides the creation of connections in the network. This is done through requiring that the links among nodes satisfy certain conditions that depend on independent parameters, like the identifier assigned to each node by the overlay network. Throughout this process, the overlay network acquires a topology that adheres to the fixed structure and is independent of the underlying physical topology.

These structures are designed to provide a set of desirable properties in the formed networks. The most common structured overlay networks fall into the category of Distributed Hash Tables (DHT). These networks are designed to store objects associated to a key, like an ordinary hash table, but in a distributed manner, assigning to each node the responsibility over a range of keys. The main concern in these systems is scalability, and its performance is measured in terms of how the cost of operations or overhead of network maintenance varies with increasing size of the network. Some of the desirable characteristics for these systems are achieving a low, bounded look-up latency (i.e., the time required to contact the node responsible for a key) and having a small routing table at each peer.

DHTs have led to the abstraction of Key-Based Routing (KBR) [Delmastro et al. 2006] which refers to the functionality of finding a node responsible for a key. This concept is useful as it allows the separation of the routing part of DHTs from their specific functionality of storing objects, which makes the overlay networks more adaptable to implement many kinds of systems with different functionalities.

There are different structured topologies proposed in the literature. Most of them provide a KBR functionality. Next we present a brief overview of the most common ones.

Ring. Overlay networks with a ring topology assign a numeric identifier to each node and identifiers are ordered in a circle. The number of nodes in the network is expected to be much lower than the number of possible identifiers, and therefore the population of the ring is expected to be sparse. Within this circle, each node, depending on the identifier that has been assigned to it and on the identifiers closer to his that are assigned to other nodes, is held responsible for a range of keys. Messages are routed using KBR, and in the case of DHTs, the node responsible for a range stores all the objects associated to a key in that range.

In order to route messages to the required key, each node stores a routing table containing the addresses of a set of other nodes. The requirements that the entries in this table must follow depend on the specific overlay network, but the usual condition is that they allow messages to be sent to different parts of the ring, in a way that can guarantee that messages will be routed to the node responsible for a key in a low, bounded number of hops.

The identifiers are usually generated randomly or by hashing some data related to the entity that requires the key. For example, identifiers for a node can be generated by hashing its IP address, while identifiers for objects might be generated by hashing their name or their content.

Examples of ring-based overlay networks are Chord [Stoica et al. 2003] and Pastry [Rowstron and Druschel 2001].

Multidimensional space. Overlay networks like CAN (Content Addressable Network) [Ratnasamy et al. 2001] place nodes in a D-dimensional space. The identifier assigned to each node is composed of D values, each inside a range specified for the corresponding dimension of the D defined. This identifier corresponds to a point on the D-dimensional space. Each node has a zone of the identifier space assigned to it. In a DHT, objects also have the same kind of identifiers assigned, and are stored by the node responsible of the zone to which they belong.

The identifiers to the nodes and objects in a D-dimensional space network might be assigned by different methods. They can be generated randomly, but the multidimensionality of identifiers can also be leveraged to address entities according to some criteria like a set of properties. For example, a set of D attributes of nodes can be defined, and the identifier of a node generated by its values on each attribute, for example, CPU speed, RAM, disk space, etc.

Tree. Some overlay networks have the structure of a tree, where each node has a link to its parents as well as one to each of its children. There are a large number of different kinds of trees with different characteristics and properties in the field of data structures, and many can be directly translated into an overlay network. Overlay networks that use this structure leverage the properties of a specific type of tree to achieve the desired behavior of the system.

However, there are a few problems inherent to trees when translating them to distributed systems. The most obvious is the fact that the root of the tree becomes a point of centralization. Therefore, it can become a bottleneck or a single point of failure, hampering the performance of the network. This requires designers to develop workarounds to avoid starting searches on the root and overloading it. However, some of the common functionalities that a tree can perform, like information spread and aggregation, are highly desirable in some systems. Because of these, sometimes trees are used together with a different overlay network to leverage the advantages of both.

One example of an overlay network with a tree structure is Willow [Van Renesse and Bozdog 2004], which provides DHT functionality as well as publish/subscribe information dissemination and data aggregation.

3.1.2. Unstructured. Unstructured overlay networks do not have a predefined topology to shape the network after. They do not impose structural limitations on the creation of links among nodes, related to node identifiers or other fixed criteria independent of the underlying physical topology. They often create random connections, although some heuristics related to the specific functionality of the application or other factors can be used to influence node connectivity. Examples of unstructured overlay networks are FastTrack/KaZaA [Kazaa 2013], FreeNet [Clarke et al. 1999], and Gnutella [2013].

3.2. Emergent Underlying Topology

Independently of whether they have a predefined structure or not, overlay networks can be formed in a way that allows characteristics of the underlying physical network, like distance among nodes, to surface and influence the formation of an emergent topology. This is usually related to trying to reduce the cost of communication by connecting in the overlay nodes that are physically near, or nodes with similar characteristics.

There are many examples of systems which allow the underlying topology to emerge to a certain extent in its overlay network, be it structured or unstructured. Pastry

[Rowstron and Druschel 2001] achieves a good locality by influencing its selection of nodes into its routing table by a proximity metric, in addition to the protocol's structural requirements. Other systems might let emerge connections related to specific application metrics, like for example connecting nodes with similar interests.

3.3. Degree of Centralization

There is one specific kind of centralization that is used in many of the systems analyzed in this article, which we call *local centralization*. We say that a system is locally centralized when only a subset of the nodes that compose it are part of the overlay network, serving nearby nodes and providing a local point of centralization. The use of these usually called supernodes or superpeers is frequently related to achieving a desired scalability with respect to performance metrics such as response time. This is due to the fact that the nodes served by a supernode do not participate in the protocols of the overlay network, and therefore its size remains small and the amount of communications and lookup latency are kept low. However, the basic problems of centralization are not absent here. Supernodes need to be sufficiently well-provisioned to perform the functions of a centralized server for their depending nodes, and although there is no single point of failure in the system, each supernode can become a single point of failure for its users. Solutions to these problems that adapt to each environment can be found, however, like configuring the number and distribution of supernodes to avoid overloading them, and have back-up supernodes to avoid the local single point of failure.

The characteristics of the specific topology of a system and the presence or absence of local centralization will determine its overall degree of centralization. Systems with the maximum degree of centralization, which would consist of a single server to provide the resource discovery to all the clients and resources, are not considered in this survey. However, not all of the presented systems, which are considered decentralized, can be said to present no centralization at all. Tree structures have a natural point of centralization in their root, and hierarchical networks too. The use of rendezvous points to store information also creates centralization in the system. Different degrees of centralization can be used in a careful trade-off to achieve various levels of desired properties like scalability, fault tolerance, and load balancing.

3.4. Information Spread Method

Information can be spread throughout a network by a set of different methods. The most traditional classification of these methods divides them in two categories: pull and push. Pull methods have the node that needs the information initiate the process of receiving it. It usually sends a query that is forwarded through the network until it reaches a node that can satisfy it. Then, this node sends the required information to the initiator of the search. Therefore, to obtain information one node needs to reach it and pull it to itself. Push methods, on the other hand, have the node that has the information send it to those who may need it. For example, a node can send information about its status periodically to the nodes that might be interested. Therefore, the node holding the information has to push it into the other nodes.

In many systems, however, both approaches are used complementing each other. It is usual in resource discovery systems to have nodes sending what is usually called advertisements, containing a specification of their capacities, which are stored by one or more nodes of the system. Nodes that look for resources of a certain type issue a query that contains their requirements. Queries are forwarded to nodes that store advertisements for resources that can fulfill them. In this way, both pull and push methods are applied into a single information spread mechanism.

The method of information spread chosen, or the specific combination of different methods, has a great influence on aspects like the performance and overhead of the system. Many different mechanisms can be implemented, but we keep our classification at this generic level of pull and push methods.

4. RESOURCE DISCOVERY MECHANISMS AND SYSTEMS

In this section, we discuss a number of resource discovery mechanisms and systems. We classify them according to their main topology, and discuss all the other aspects mentioned before, both requirements and design characteristics.

Those systems that have a name given in the literature describing them will be identified by such name. Systems for which no name is given will be identified by the author name and year of the first paper in which they are described.

4.1. Structured

In this section we present the systems that are built using a structured network. Many of them use existing DHTs, either to augment it or to build a system over it, while others present their own architecture. They are further categorized in the aforementioned different types of structured networks. However, some of them do not fit in one of these categories, either because they use a different type of structured network or because they use more than one of the known types in their implementation. These systems are filed under the category of *hybrids and others*.

Ring

4.1.1. Ranjan et al. 2007. The system presented by Ranjan et al. [2007] is modeled like a grid where each site has a dedicated server for resource discovery. This server acts as a broker who resolves queries locally in a centralized way, and also connects the site to the other organizations of the grid. This node is part of a DHT, where it publishes the computational resources of its site, and can contact the brokers of other sites. Specifically, the system is implemented over Chord.

Through a decentralized publish/subscribe system, the users who need resources subscribe to the kind of resources they need. Available resources periodically issue events to advertise their availability. The resource descriptions are mapped to a d-dimensional space, which in turn is converted to a one-dimensional value. The space is divided in cells, and each cell is converted into a value that is hashed and stored into the DHT. Subscriptions are issued by nodes needing resources, while the events are resource ads. Both are pushed to the nodes responsible for the cells.

Queries are composed of d attributes. They can specify a single value for each attribute, or ranges of values for some or all attributes. Range queries are stored diagonally in the d-dimensional space, instead of covering the whole area, which could potentially belong to more cells.

The system is said to be scalable, and according to the authors there is no significant impact of system size on performance (query latency). The presented tests use group sizes of up to 500 nodes. However, a high level of churn can make Chord perform badly, with a huge overhead. Their simulations showed that, with a limited message queue on nodes, there was an important degree of message loss, preventing many nodes from being found and queries from being answered since their corresponding advertisements and subscriptions were dropped. Regarding security, since there are no replication methods incorporated, any malicious node would get control over a part of the range of values, therefore being able to prevent access to nodes falling into that range.

4.1.2. Chang-Yen et al. 2008. The system presented by Chang-Yen et al. [2008] uses a Pastry overlay network for each considered attribute. Each node that offers a value for it above a certain threshold (e.g., a certain amount of disk, etc.) belongs to the corresponding overlay. Nodes that belong to more than one layer are called gateway nodes, and allow queries to pass among layers. Each node also stores a local cache with the definition of nearby resources in the layer or layers it belongs to. They periodically send their data to nearby nodes (a subset of the first row of Pastry routing table) and also, more infrequently, to farther nodes.

Whenever a node needs to locate resources, it first checks its local cache to know if there are any nearby nodes that can answer the query. If there are not, the query is forwarded to a layer that the node does not belong to and that can provide adequate values for a required attribute. In order to reach the target layer, the query is forwarded towards a designated gateway, located in a well-known Pastry key. Therefore, the preferred method for information dissemination is by pushing it to nearby nodes, while farther information can be pulled when required. This method does not provide completeness, and requires that great amounts of resources satisfying requirements are available for queries to complete.

Since the method for looking up resources is based on caching information about nearby nodes, and the routing towards designated gateways to reach different layers is based upon Pastry, it is scalable. However, tests only show networks sizes of 60 nodes.

Resources are defined by a fixed number of attributes, with a layer per attribute in the system. The queries only search values greater or lower than a given value. However, it is said that with a modification to create layers for intervals of values, instead of values above a threshold, range queries could be supported.

Its fault tolerance is based upon Pastry protocols and replication of the designated gateways. It is also relatively resilient to attacks, since nodes cannot prevent other nodes from sending their advertisements to neighbors. The multiplicity of gateways among layers increases the probability of reaching the desired layer for a query, but a faulty node can forward queries to layers formed entirely by malicious users.

4.1.3. XenoSearch. XenoSearch [Spence and Harris 2003] is built to search for resources that match a description that can be complex, containing range attributes, locality properties regarding one or more nodes, etc. The system is composed of a Pastry network for each attribute/dimension. Each of them has a set of Aggregation Points (AP) which work together to aggregate the information of a part of the network. This aggregation structure forms a binary tree, with an AP for the whole network, one for each half of it, and recursively. The APs are located by the Pastry routing function, assigning them to a specific key obtained through the division of the identifier space. These APs contain bloom filters that indicate what nodes are in the network and can be easily aggregated.

Nodes into the overlay periodically send a message to their assigned AP to update their state. This AP forwards its aggregated information up the tree, making it so that each AP can answer queries regarding the range that it covers by making a summary of results using bloom filters. However, the use of bloom filters means that, while all resources fulfilling a specific set of requirements can be found, XenoSearch can return false positives. Therefore, the final responsibility on choosing a node is on the client, who has to check the returned list of nodes to identify those that actually satisfy all the requirements. Moreover, only queries that can be expressed as a range of acceptable values for a predefined set of different attributes, and OR and AND operators, are considered. The client must issue range queries in each dimension and unite or intersect the results as needed.

The architecture of XenoSearch uses local centralization, where nodes in the overlay are only a subset of the network. These nodes offer the resource discovery service, and other nodes use it to publish their resources and to obtain them when needed. The overlay network, however, is decentralized. Each node derives its id in each overlay from its value for the corresponding attribute. However, malicious nodes can make up false identifiers to gain control over some APs, making some nodes impossible to locate.

Nothing is mentioned about the scalability and fault tolerance of XenoSearch. Since it uses Pastry, it can leverage its properties, but the presence of a number of Pastry rings can make the overhead of stabilization protocols and repairs costlier. Moreover, periodical operations for maintaining trees are required for each dimension. The tests showed in the paper only use 18 nodes.

4.1.4. Abdullah et al. [2009a, 2009b]. The system presented by Abdullah et al. [2009a, 2009b] is formed by three kinds of agents: producers, which execute jobs on behalf of other users; consumers, which use the resources of the network to execute jobs; and matchmakers, which match queries issued by consumers with ads of producers that can satisfy them. Nodes in the network can simultaneously perform the functions of the three kinds of agents. Moreover, nodes can be dynamically promoted to matchmakers when the existing ones are heavily loaded. Once one is promoted, it remains acting as a matchmaker until its load decreases, the moment the when it quits being a matchmaker.

The load of the matchmaker is measured by a function called TCost, which calculates the cost of each request counting the number of preceding requests to be processed before that one. When average TCost goes over a threshold value, a new matchmaker is promoted. Therefore, the system automatically balances the load on the matchmakers, considering that the query frequency distribution is uniform among the DHT. However, these matchmakers are a point of centralization.

The topology of the overlay network is a circular DHT, Pastry. The id space of the DHT is divided into zones. There is a total division of the maximum of N nodes admitted by the DHT (depending on the number of bits of the identifiers) on M zones. For each of these zones there is a designated matchmaker, which corresponds to the first node of the next zone. In the beginning, only one of the matchmakers is activated. The others are only activated when the contiguous matchmaker is overloaded. When the number of active matchmakers is lower than the number of zones, zones are joined and covered by one matchmaker. When there are less than M zones and matchmakers are promoted, zones with more than N/M nodes are divided. This makes the system scalable, although only resources on the same zone of the DHT can be found, and no failure tolerance is considered.

Providers send their specification to the matchmaker assigned to their zone, codified in the bidding language used by the matchmaker in what is called an ask. Consumers requiring resources send their bids, containing their requirements about time, price, and amount of resources, to the matchmaker. Matchmakers use a continuous double auction to match bids with asks and answer them when they are received. The bidding language used in the auction can provide a high flexibility in asking for resources, but requires a higher intelligence into the clients to calculate pricing and bidding strategies. The auction does not provide completeness in results, even considering only the zone assigned, but does provide accuracy.

The authors present experimental results obtained with up to 650 nodes and 5 matchmakers. However, only the temporal evolution of metrics like TCost and response time while matchmakers are promoted and demoted is shown.

4.1.5. Abdullah and Bertels [2011]. Abdullah and Bertels [2011] present a system which uses nature-inspired self-organization mechanisms to improve the efficiency of an auction-based resource discovery mechanism. Specifically, it uses ant colony optimization to route bids for resources towards fitting matchmakers.

The system is composed of a number of matchmakers, each associated to a category of resources. They also have a pheromone value computed from the number of matches they perform. Each client node is assigned to a matchmaker and sends it resource requests or offers.

When a matchmaker receives a bid, it checks if its associated category matches that of the bid and if its pheromone value is higher than that of neighbors also associated to that category. If this is the case, the matchmaker tries to perform a matching between the requests and offers it stores through an auction mechanism. Otherwise, it forwards the bid to the neighbor with the highest pheromone value among those with the same category. In case neither the matchmaker nor its neighbors are assigned to the category of the bid, it is forwarded randomly until a fitting matchmaker is found. When a match happens, the node that sent the bid becomes assigned to the matchmaker that performed it.

The matchmaker network is built on top of a Pastry overlay, which allows nodes to discover matchmakers in case of failure. This provides a high fault tolerance. The self-organizing mechanism provides scalability, since nodes become assigned to the matchmaker with the best probability of finding a match, forming an emergent underlying topology. However, in the worst case a bid would be routed to all the matchmakers. Security is not mentioned in the paper, but if no additional mechanisms are implemented, a malicious matchmaker could trick its neighbors into believing it has a high pheromone value and eventually gain control of the matching of a large amount of bids.

4.1.6. SWORD. SWORD [Oppenheimer et al. 2005] proposes, first, a language to specify the set of resources required by an application, formalized in an XML that includes minimum and preferred attribute values per node, with penalizations for values between the minimum and the preferred and relations among nodes of a group and among groups, basically the latency. As can be seen, this language has an extreme flexibility and allows a high variety of constrains to be specified into a query.

Second, the SWORD paper proposes a set of mechanisms to search for resources on a distributed community. Three of them use Bamboo, a ring DHT based on Pastry, where a pair <attribute-value> is assigned to a key. The first mechanism is called SingleQuery. It requires that each node advertises itself, including the values it has for all its attributes, on every node of the DHT assigned to each of its values of the attributes. Queries only require a message to the responsible of a key, or more precisely, to a range of keys corresponding to the range of values that fulfill the requirements. The node responsible must process the query and search for nodes whose specifications are stored locally and satisfy the requirements. The second method is MultiQuery, where each node sends only the value of an attribute to the node that stores ads for that attribute and value. Queries require contacting all nodes that store some desired value for an attribute, and the results must be compared at the querying node to find which nodes have all the required values for all attributes. The third is Fixed, which requires a set of infrastructure servers. In this mechanism, each node sends its resource description to one server chosen randomly among the predefined set, and queries are sent to one of those servers and redirected to all the others. Alternatively, ads can be sent to all servers and queries to only one. The last mechanism is Index, which combines the SingleQuery and Fixed approaches. In this mechanism, there are a set of infrastructure servers which keep a mapping of value ranges to DHT nodes that manage them. Updates are handled as with the SingleQuery approach, while queries are sent to the index server responsible for the key range of interest, that forwards the query directly to the DHT nodes responsible of that range instead of using the DHT routing.

In order to find nodes that satisfy the desired inter-node and inter-group requirements, a separate mechanism is used, where nodes are clustered and representatives are used for each cluster. These representatives measure their latency with all other representatives, and each node has as a part of its specification a link to its representative. Once nodes that satisfy individual requirements are retrieved, the representatives are contacted in order to compute the inter-node requirements. Therefore, this mechanism is compatible with any resource discovery mechanism and can be added independently to any of the four proposed approaches, or to alternative ones. However, the mechanisms for the creation, selection, and monitoring of the representative nodes are not explained. Finally, SWORD implements an optimization component which computes the possible combinations among the nodes returned by the distributed search and the information from the representatives in order to find the groups that best suit the requirements. This is executed locally, so can also be added as a local module with any resource discovery mechanism that is used.

Fault tolerance is high for the distributed approaches, since Bamboo is a DHT designed to tolerate churn. Fixed and Index mechanisms, on the other hand, depend on the number and reliability of infrastructure servers. The security of the decentralized mechanisms is high since nodes register in a number of nodes, and therefore cannot be hidden by a single malicious node.

The centralized version of SWORD is deployed and running in PlanetLab [Peterson et al. 2006; PlanetLab 2013]. Queries for resources can be submitted via Web [SWORD] using the attributes monitored by CoTop [CoToP], a monitoring tool for PlanetLab. However, its inter-node requirements' evaluation mechanism is not implemented in it.

4.1.7. Cheema et al. [2005]. The system presented by Cheema et al. [2005] uses Pastry to connect the nodes of the network. These nodes have identifiers that are generated, randomly or otherwise, by the node itself, therefore allowing for malicious nodes to occupy a specific place in the network.

The characteristics of the resources are encoded into a string, including a static part, conformed of CPU speed, RAM, disk, and OS configuration of the node, as well as a dynamic part that contains the current CPU, RAM, and disk-free percentage. Therefore, resources are not assigned to a point in the space, but to an arc, from the minimum to maximum values of the dynamic part, bounded by the static part. Identifiers are assigned by hashing the static part of their description and then substituting the lower 32 bits by those that represent the dynamic part.

The information about the resources is stored by the nearest node in the identifier space, as is the normal behavior of the DHT. Resources must issue an update when their dynamic data changes so that their description is correctly stored. However, this might lead to stale data, since the update process changes the resource id and can therefore change the node responsible for the information, leaving invalid data in the former one. To avoid this, updates can be issued periodically, allowing nodes to delete the information that has not been renewed in a specific period of time.

In order to find a certain type of resource, a node must issue a query and forward it through the DHT to the identifier that encodes the desired resources. However, queries require exact matching in static attributes in order to find and answer. To improve this, the system allows for iterative queries varying dynamic attribute values. This allows querying for a range of values for the dynamic attributes, but static attributes, which define a node's capacities, must be fixed in the query. Therefore, no real range queries are supported by the system.

The scalability of the system is that of Pastry, since both advertisement messages, including insertions and updates, and query messages must be routed through the DHT towards the node responsible for the identifier. As stated, however, performing a range query would require a large amount of iterative queries and therefore be much less efficient. The tests presented, using a network of 1000 nodes, do not show the number of messages required for queries, but only overhead measures like the total number of messages in the system. The fault tolerance is also that of Pastry, introducing only a small delay after a node fails until periodical updates bring the system's information up to date again. Finally, security is not considered in the system, and since it is explicitly allowed for a node to create several virtual nodes and insert them into the network with self-generated identifiers, a malicious node could easily take over a great part of the system, rendering it unusable or allowing only access to compromised resources.

4.1.8. Squid. Squid [Schmidt and Parashar 2004, 2008] distributes objects, whether resource descriptions or data items, in a multidimensional space, according to the values associated to a set of attributes. As a difference with other systems that take a similar approach, Squid allows the existence of keywords for describing such items. This is done by allowing values in an axis of the multidimensional space be formed by any set of characters and ordering them lexicographically.

A fixed number of axis, and therefore keywords or attributes, is set to describe objects. This multidimensional space is then converted by using Hilbert's space-filling curves into a unidimensional space. Nodes in a Chord DHT, whose ids are assigned randomly, store these objects, having a portion of the space (a cluster of the multidimensional space) assigned to them. Since the clustering process is recursive, so are searches, with queries sent to a node responsible for the highest-order cluster being processed and directed to the node responsible of the fitting lower-level cluster recursively.

Queries can specify exact values or ranges, as well as wildcards for keywords, that is, the user can specify the initial part of a word, or query for any word. They use a query optimization engine to visit the minimum number of nodes possible during the recursive process, which has a lower bound in the number of nodes that store the desired data.

Tests with up to 5400 nodes show the relation among number of matches, data nodes, and processing nodes of a query. For example, as the authors state, solving a query with 160 matches can be more costly than solving one with 2600 matches.

The fault tolerance of the system is that of Chord. Security is not considered in the paper, and since node ids are generated randomly, it would be easy for a malicious node to choose an identifier for itself that would give it control over a specific cluster.

4.1.9. DGRID. DGRID [March et al. 2007] is a system for service discovery in grids composed by resources owned by several administrative domains. It categorizes resources in types which are assigned to a specific identifier and mapped to a Chord network. However, instead of sending advertisements to the corresponding node in the network, DGRID uses a virtualization scheme to create several nodes for the same type of resource into a network, one for each domain. Each node is the resource itself being published, and by adding specific entries in the finger table of nodes that point to other nodes of the same domain, DGRID ensures that failures in a domain will not affect availability on other domains.

The overhead of maintaining the DGRID network is greater than that of an ordinary Chord, since each node must maintain a greater number of entries in its finger table. However, this allows selective proximity-based or domain-based routing, and also offers a greater fault tolerance, since the existence of alternative paths makes it difficult for a failed node to cause a query to fail. Moreover, if security into an administrative domain is tight, its nodes can use the neighbors in their own domain for the routing without the possibility of interference by malicious nodes. The fact that resource information is only stored by the very resources also increases security and decreases overhead. However, this comes at the cost of the extremely limited flexibility for defining resources and queries.

Their tests use a network formed by 25000 administration domains, with a variable number of resource types per domain. The results show that the average hop count is between 4 and 7 approximately.

4.1.10. MAAN. The Multi-Attribute Addressable Network (MAAN) [Cai et al. 2003] uses Chord to create an identifier space populated by nodes. However, instead of using a common hashing algorithm like SHA-1 [2013], it uses a uniform locality-preserving hashing technique. This guarantees that consecutive values will be located consecutively in the DHT after applying the hash function. In order to index resources described by a fixed number of attributes, MAAN requires them to register once for each attribute, hashing the pair <attribute, value> and storing the resource description in the corresponding node. Queries can be sent to the node responsible for the value assigned to one of the attributes, which is considered dominant for the query, and processed there to extract the list of resources that fulfill all the attribute requirements. Determining which of the involved attributes is the dominant one is left for the issuer of the query.

Range queries leverage the use of the locality-preserving hashing by starting at the lower value of the range. From there, they are sent to the next consecutive node, sequentially until the last node responsible for the range is reached. In each of these nodes, the query is processed and the results are returned to the client. Fault tolerance and security are inherited directly from the DHT, with failures requiring a recovery process and malicious nodes being able to control parts of the identifier space.

The authors present tests using a network of up to 128 nodes where the number of routing hops required for range queries is shown. With exact match queries the number is logarithmic to network size, while with a range selectivity of 10%, it becomes linear. The query cost is also presented as a function of the range selectivity. The cost is 10 hops for a 10% selectivity, while for a 90% it increases to cover all the nodes in a network of 64 nodes. They also show that, with a 10% range selectivity and 64 nodes, queries dominated by a single attribute require about 10 hops, while when using an iterative search the number of hops grows linearly with the number of attributes in the query.

4.1.11. Self-Chord. The paper by Forestiero et al. [2010] presents Self-Chord, a bio-inspired algorithm which ads self-organization and load balancing to a Chord overlay network. It does so by decoupling node ids from the contents they store.

Nodes in Self-Chord have an id and stored objects have a key, but, unlike in Chord, these do not need to be in the same range. Nodes are connected forming a ring, ordered by their id. However, objects can be stored in any node independently from their key or the node's id. A set of agents traveling throughout the network are responsible for moving and organizing objects.

These agents randomly take an object in a node with them to the next node. The probability of taking a node is inversely proportional to the similarity of the object's key with the node's centroid (i.e., average of the keys stored in the node). This way, nodes have a high probability of storing objects similar to each other. This allows users to find objects with an efficiency similar to Chord, (i.e., logarithmic). The advantage is that load balancing is performed automatically and communication cost does not depend so strongly on churn as in Chord, since there are no costly recovery mechanisms which involve moving objects according to a deterministic scheme.

Self-Chord could be used for single-attribute resource discovery by storing ads for each resource as objects. This would require periodical republishing for failure tolerance.

Security is not commented in the paper, but malicious nodes with the ability to stop agents going through them could severely impact the self-organization process.

Multidimensional space

4.1.12. Kim et al. [2007]. The papers by Kim et al. [2007; 2008] define a model where users provide their resources to execute jobs. The matchmaking is totally distributed and decentralized. It uses a CAN to organize the peers according to their offered resources.

A CAN is where the point in the d-dimensional space of each node is determined by its characteristics, that is, the value assigned to each of the d attributes that define resources. A job description, including information on how to retrieve the files and execute the job, is sent towards the resources it requires by using the CAN routing. Once there, the node, called owner node, that received the query must push the job towards a node that has an empty (or lightly loaded) queue.

The system aggregates various types of information including number of nodes, length of queue, etc. The aggregation is done from nodes that have more resources, that is, a higher value in each dimension, towards the lower ones. This information must be updated periodically. The information is aggregated in a dimension, but also using nodes that are near in the other dimensions. For example, with 3 dimensions, a node (1,1,1) would receive, aggregating only in the first dimension, information from (2,1,1), (3,1,1)...(N,1,1). But they aggregate too, with a modifier based on distance, information about (2-N,0,1), (2-N,2,1), etc. This information is used to load the balance by sending the jobs to the nodes that have a shorter queue, and stop pushing them once it is determined, based on local information, that the best node has been reached.

It is assumed that a node can only execute a task at a time, and nodes have a queue of jobs to execute. They are sandboxed in a very simple way, using chroot to deny them access to the file system, although the use of virtual machines is suggested.

Queries require a minimum, or exact, value in d attributes that are mapped into a d-dimensional space. Closed range queries are not allowed.

The fault tolerance of the system is that of CAN. The results in the paper show that the recovery process is complex, and in case of dynamic nodes (between 10% and 30% of the nodes leave) the wait time for jobs increases. The scalability of the system is also hampered by the complexity of maintenance processes, but they propose some ways to improve it. However, the presented tests do not specify the network size simulated, so there is no precise data about the scalability of the system.

Malicious nodes can control nodes for which they are owners. Furthermore, they can stop pushing jobs that arrive to them, as well as providing false aggregated information to influence the route of pushed jobs. The influence of these maneuvers in the overall system will depend on the distribution of resources' characteristics and jobs' requirements, which will determine the ability of a node to position itself in a highly influential point in the d-dimensional space.

4.1.13. Costa et al. [2009]. The paper by Costa et al. [2009] presents a system where nodes are arranged in a multidimensional space according to their definition in a set of <attribute, value> pairs, assuming that the number of attributes is fixed and that their values can be uniquely mapped to natural numbers. The space is divided into smaller spaces hierarchically, called cells, with various levels of subdivisions, that is, each cell contains a number of smaller cells which may in turn be divided in more cells. Each node has a set of links with nodes in the neighboring spaces in each level, and these links are used to forward queries towards the nodes that can fulfill them. In order to accommodate additional or rapidly changing attributes, these should not be mapped as dimensions, but only be checked locally by each node while the query is forwarded, directed by the more static attributes. Queries are forwarded until the required number of nodes is found, and then the path of the query is followed back to the user by the response.

In order to maintain the overlay network in the presence of failures, the system uses a two-layered gossip-based protocol. The lower layer uses an unstructured overlay with random connections, while the upper layer is structured by attribute values and linking nodes with others in neighboring zones. The lower layer is used to maintain the upper layer connected in presence of churn by providing random links.

The evaluation of the system shows that no message has ever been received twice by the same node, and that in experiments where the system does not experience churn, completeness is achieved. The overhead caused by the overlay maintenance is not considered in the results. The routing overhead is measured in number of hops on nodes that do not match the query, which remains in average under 3 for network sizes of up to 100000 nodes. However, when there is a small proportion of the nodes in the system that can satisfy a query, the routing overhead can increase up to almost 300 hops in the worst case with a network of 100000 nodes and requiring complete results.

The number of links that each node must maintain is a function of the number of dimensions and the number of levels, although the real number of neighbors might be smaller since cells that are empty do not have an associated link. However, these two factors do not depend on system size, so the number of neighbors has a constant limit, albeit potentially large, for a given system, regardless of variations in size due to churn or other factors.

Since alternative paths to a node might be possible and the neighbors are changed through the random gossiping of the lower layer of the network, it seems like malicious nodes could not hamper the functionality of a large portion of the system. However, nothing is stated about the security of the system.

4.1.14. MURK. MURK [Ganesan et al. 2004] is a system to support multidimensional range queries in p2p environments. In this system, the multidimensional space is divided whenever a node enters the network, much like in CAN. However, unlike CAN, MURK does not split the space equally, but divides it instead trying to split the *load* associated to that space equally. The network is connected by links between neighbor nodes in the space, as well as the multidimensional equivalent to skip pointers [Aspnes and Shah 2003], which allow nodes to contact peers at exponentially increasing distances from itself. Two methods for establishing these skip pointers are presented. The first one is random, which uses random walks on the overlay network. The second one uses space-filling curves to approximate the distance between nodes in the space.

MURK can be used as a resource discovery system based in advertisements using a pull/push approach. This approach brings potential security threats, as it gives nodes control over the access to resources in a specific range. However, its accuracy is high, as well as its efficiency in stable environments. Nevertheless, load balancing in MURK under dynamic conditions is difficult and might be highly expensive.

The authors present some tests using networks of up to 200000 nodes with two-dimensional data, achieving a routing cost of under 15 hops. Data locality, that is, the number of nodes that store the answer to a query, is very good according to the presented data, with queries almost always hitting just one node.

Tree

4.1.15. NodeWiz. NodeWiz [Basu et al. 2005] is based on a tree structure that is used to store advertisements spread by resources available in the network as a distributed decision tree. The descriptions of nodes are formed by pairs <attribute, value>. The tree divides the resources at each non-leaf node according to the value of an attribute. That is, at each node, a value of an attribute is specified and resources with a value for that attribute lower than the one specified go in the left branch, while those with a greater value go in the right branch. When a node joins, it shares the load of the most loaded node, in order to attain a good load balancing. The division made at each node is decided through a clustering algorithm over the ads stored, in order to perform the most advantageous division.

Queries sent by users are routed from the root of the tree towards the nodes that store ads that can satisfy the queries. Range queries are allowed, but the number of nodes that the query has to visit increases as the query becomes less specific, that is, all values within a range, or even any value, are accepted.

There is a single point of entrance to the tree, the root, which might become overloaded and hamper the fault tolerance of the system. The paper presents an optimization that allows caching of subtrees on the nodes that issues queries, using information returned together with the information about the found nodes. This information can be used to accelerate frequent queries, by accessing directly to a subtree that can satisfy the query without having to contact the root.

The tests show that NodeWiz can scale up to 10000 nodes with a slow increase in hops (about 5 in average). It provides complete and accurate results, although the effect of failures is not considered since NodeWiz is thought to be deployed on stable infrastructure nodes. Therefore, only ordered departures of nodes are managed.

Load balance is maintained throughout the lifetime of the system by having underloaded nodes leave and rejoin the system. Since the joining process has the new node dividing the load with the most loaded node, it attains load balancing dynamically. However, the balance of the tree is not considered, and nonuniform distributions can result in highly unbalanced trees.

Security is not considered throughout the paper. A malicious node could produce great imbalance in the tree by giving false values regarding its load, thus making all joining nodes share their load with it, while other nodes have a higher load, and making the path to the new nodes longer. Also, if it joins soon enough to be in a high part of the tree, it can direct many queries to other malicious nodes.

Hybrids and Others

4.1.16. NR-Tree. Network-R-Tree (NR-Tree) [Liu et al. 2005] proposes a mechanism for resolving complex search queries in p2p systems by adapting the spatial index R*-tree [Beckmann et al. 1990] to a peer-to-peer environment. Items are located in leaf nodes, which are passive peers that store the data indexed in a local R*-tree. Each passive node is assigned a location in a multidimensional space according to the data they store. The nodes are grouped by their multidimensional coordinates, using Minimum Bounding Rectangles (MBR). This is done recursively, grouping rectangles into bigger rectangles at each level, until the root tree.

There is an overlay network formed by superpeers, having a few per cluster to provide fault tolerance. These clusters are formed by nodes holding semantically close data, which is measured by the centroid of data in the multidimensional space. In the case of resource discovery, the space maps the attributes that define resources. Since superpeers have an identifier codified as multidimensional data, the overlay network uses a routing system similar to CAN. Therefore, for processing a query, a superpeer, which acts as a server to the peers of its cluster, first routes it to the node responsible for the adequate portion of the space. Once in the cluster, the superpeer uses its NR-tree, that stores partial information of the peers of the cluster, to forward the query to the nodes that can find the items that satisfy it.

Superpeers are elected by existing peers based on the computational capabilities and bandwidth. They are replicated and chosen dynamically, and queries are sent to backup superpeers that start processing them if no cancellation is sent in a specific period of time. This mechanism helps provide fault tolerance and supports churn. With two backup superpeers per cluster, with a 40% failure rate of superpeers, the failure rate of queries is below 5%.

Range queries are routed through rectangles intersecting with the search space, and k-Nearest Neighbor (kNN) queries, which require the k nodes that are nearer to a specified point, are routed in a depth-first way, selecting in the first route the k better nodes and then backtracking through the branches that offer a better distance. The number of attributes that define resources is fixed, as it matches the number of dimensions of the routing space.

The system is designed to store objects in each node, and partial information is pushed towards supernodes. With this information, superpeers can forward queries to the adequate nodes. This makes the system scalable. The system is also capable of providing complete answers, although it can also provide approximate answers with reduced cost. With a TTL in inter-cluster propagation the cost is reduced and responses can be adequate with a fitting distribution of resources. In the presented tests, kNN queries obtain 100% of results with a TTL of 2, while range queries need a TTL of 3. There is also a trade-off between the number of clusters and cluster size, since they affect both query performance and fault tolerance.

There are some aspects that are not treated in NR-Tree, since the system is not specifically designed for resource discovery. One of them is the publication of resource information. If each node forms part of the network and only provides its own description, then having local R*-Trees does not make much sense. If the passive peers act as supernodes of their own local cluster, indexing a set of resources, then there is the question of how the updates of the resources are managed, since the join and departure of resources would change the centroid of the data stored by a passive peer.

Malicious peers in the network could affect the routing through the multidimensional space, as well as provide false information about the nodes in their cluster, as long as they are superpeers. The algorithm for superpeer election is not specified in the paper, although it is said that they are chosen according to their computational capacities. If the algorithm uses also a trust or reputation management system, it can increase the probability that malicious peers will be excluded from the overlay network.

4.1.17. Cone. Cone [Bhagwan et al. 2004] builds a heap to order resources according to a key value. This key value corresponds to an attribute of the node, like CPU speed or RAM. Specifically, nodes form a trie, where each non-leaf node is decided based on an aggregation function over the key value. In the paper they present the mechanisms needed to use Cone with Max (or greater than) operator, where each node is the one with greater value between its two children. That means that if a node is in the higher level of the tree, it is also in all the other levels. In order to manage node joins and load balancing, it uses a DHT. Nodes have a random identifier assigned and a corresponding position in this DHT. The DHT acts as the lowest level of the trie, thus inserting nodes into a random place and helping maintain the balance of the trie. Cone is implemented using Chord, but it can be built over any DHT that allows longest-prefix routing.

Regarding its search flexibility, Cone only allows to locate resources with a key value (i.e., the value for a certain attribute) equal or greater than a given value. This is because the presented mechanisms use the Max function over a single value. In the paper they introduce methods to improve the flexibility and allow searches for multiple attributes, by combining values of different attributes in a single key by bit-interleaving, although its ordering properties are not the same, and therefore Cone does not offer the same properties for multiple attributes that it does for one. With one attribute, it provides a total accuracy and completeness when the key values do not change very frequently, but in extreme cases its accuracy can drop substantially. More specifically, the presented experiments show a lag of 2.2 seconds when values of a node change, and the expected accuracy is 82% with values changing every 6 seconds and 93% with changes every 30 seconds.

Cone uses a pull method to spread information, with nodes issuing queries that are routed towards nodes that can satisfy them. The average number of hops for finding nodes satisfying a query is log(N), N being the number of nodes in the system. Therefore, it offers a good scalability. In the simulations presented in the paper, the system has been shown to scale to network sizes of up to 10000 nodes. It also provides mechanisms to recover from failures, which require (log N)2 messages.

It is a decentralized system, although its tree structure seems to give it an inherent point of centralization. However, their tests confirm that for a variety of query distributions, the load is fairly balanced among all nodes. Although nothing is said about security, a single node reporting a false key value could prevent a large portion of the resources in the group from being found from certain positions in the DHT. However, good load balancing should help minimize the effect of such an attack.

4.1.18. Gao and Steenkiste [2004]. In the system presented by Gao and Steenkiste [2004] a Range Search Tree (RTS) is built over Chord to index the stored resource descriptions and allow for range queries. Resources are defined by a variable number of attributes, and each resource must register in the DHT for the corresponding value for those attributes that define it. Queries must be issued using the attribute that is expected to be more relevant.

For load balancing and fault tolerance, identifiers are not assigned to a single node, but to a matrix. This is done dividing the registration to the identifier randomly into a number of partitions, which can be created on demand when the load increases, and by replicating each partition into a number of nodes. The size of the matrix is stored in the head node, the one responsible for the identifier according to Chord, and registering nodes must select randomly one of the partitions and register with each of its replicas. Queries, instead, must be sent to any replica of each of the existing partitions.

Each identifier corresponds to the hash of a certain range of values for a certain attribute. Therefore, since the RTS divides the whole range of allowed values for an attribute, contained in the root, in two subranges recursively until arriving to leaf nodes, each of these nodes has an identifier associated and can therefore be found through the DHT routing.

In order to minimize the number of messages required for registering and finding resources, the RTS can adapt to load. There is a default level where queries and registrations start to avoid overloading the root of the tree, and the rest of the tree is inactive. The nodes in this level count the number of registrations and queries they receive, and depending on the load and the range distributions, which can search ranges bigger or smaller than the one covered by this level, the active part of the tree can be expanded upwards or downwards. During this upwards or downwards process this part of the treeies to optimize the registration and query process by minimizing the number of hops required for the predominating values.

The paper presents tests with a network of 10000 peers, using different variations of the presented algorithms, and measures the number of messages involved in query and registration. First, number of messages per query is measured with an increasing registration rate, and the results show that with a registration rate of 100 registrations per second or less, the mean number of messages is lower than 2 for the best algorithm. With higher registration rates, up to 2000 registrations per second, it increases until approximately 10 for the best approach. They also present results of the number of query messages as range length increases, showing that it grows linearly when the load is high (2000 registrations per second) but remains constant when it is low (25 registrations per second).

Security is not considered in the paper, and although matrix replication seems to avoid total control over an identifier by one node, in truth the head node can control it. However, the adaptive tree makes it difficult, although not impossible, for a malicious node to gain control over specific ranges of the attribute space.

4.1.19. Tanin et al. [2007]. The system presented by Tanin et al. [2007] uses a quadtree to map resource descriptions and queries to nodes in the network. Both are considred as rectangular spaces defined by ranges of values in the attribute space. In a bidimensional space, the quadtree divides the space recursively into four square blocks. In order to process a registration or query, it is stored in a level as low as possible that doesn't require the object to be subdivided.

The quadtree blocks are hashed into the Chord identifier space and assigned to nodes in the network. Therefore, nodes can send queries to be processed to the tree by using the Chord routing mechanism. In order to avoid all queries being sent to the root of the quadtree, it is determined that they start at a specified minimum level greater than zero. In order to bound the cost of queries, a maximum level is also fixed, which determines the maximum number of DHT lookups needed. Moreover, nodes in the tree cache the addresses of their children in order to allow direct messaging from the root to the leafs of the subtree considered in the search. The starting point can be locally calculated by the node issuing the query.

Security is low, since there is no replication and a node controlling the higher levels of the tree can hold control of a great part of the network. Fault tolerance, on the other hand, is provided by Chord, but the addresses cached by quadtree nodes can disappear or become invalid in presence of churn. Therefore, the performance of the search mechanism can be notably affected.

The authors present some tests with a network of 1000 nodes that show the behavior of the system, measured by the average time and number of messages per query when parameters of the system change. The first one is the minimum level permitted, which causes both metrics to grow notably when increased over 5. Other factors like the maximum level and the query scale cause smaller increases in response time and message count. Finally, the system is tested with a network of between 500 and 3000 nodes showing no notable effect in response time. In these same experiments, the cache miss ratio of the tree links is also increased up to a 15% with a moderate effect on response time. However, the response time in any case is reported as above 40 seconds.

4.1.20. P-tree. P-tree [Crainiceanu et al. 2004] is an index structure that allows range queries in a peer-to-peer environment. It is not specifically dedicated to resource discovery. Moreover, it has little flexibility, allowing only for node id to be used for search. However, there are possibilities of using this as a base for a resource discovery system. For example, as some of the systems presented in this paper propose, multiple overlay networks can be used to index according to multiple attributes, or an aggregation of multiple attributes can be used as identifier.

The system uses Chord to organize a set of nodes in a ring according to their identifier. Each node locally stores the path from the root of the tree to itself, building the tree as if the node has the smallest value present in the network. This local information is periodically updated by a stabilization process. This allows queries to be routed to the adequate subtree in a logarithmic time. Specifically, the search cost in a P-tree of order d is $O(m+logdN)$, N being the number of peers in the network and m the number of peers in the selected range.

The results presented by the authors show that the increase in search cost is logarithmic, with tests simulating networks of up to 250000 nodes. However, the number of hops is high, going from almost 100 for a network of 1000 nodes to a maximum of about 150 hops. A real implementation with 6 nodes simulating 30 virtual peers shows a response time for searches of 0.043 seconds in stable environments, while in an inconsistent network it takes 2.796 seconds. The stabilization process itself takes 17.25 seconds.

P-tree offers a good fault tolerance, leveraging the mechanisms provided by Chord and adding some eventual consistency guarantees. It also provides completeness as well as accuracy. Moreover, the use of local information and multiple visions of the tree without a single root not only enhance performance but also security, making it difficult for a malicious node to purposefully control a range of the network.

4.1.21. SCRAP. Ganesan et al. [2004] propose SCRAP, a system where nodes are mapped into a multidimensional space which is converted into a unidimensional space using space-filling curves. After this conversion, the range is divided in subranges which are assigned to the nodes of the network. These nodes are linked in a skip graph [Aspnes and Shah 2003] which consists of a circular list and a set of skip pointers, randomly set at each node to nodes at exponentially increasing distances from itself, which allows the routing to proceed faster to the target node.

Multidimensional range queries are converted, using the space-filling curves, into a number of unidimensional range queries, as a given range in the multidimensional space may not be located contiguously in the unidimensional space. This also implies that some undesired results might be returned, including nodes that do not belong into the required range.

The authors present some tests using networks of up to 200000 nodes with two-dimensional data, achieving a routing cost of under 15 hops. However, the data locality, that is, the number of nodes that store the answer to a query, grows linearly until presented values of over 30 nodes. The total cost of answering a query is, in fact, the sum of both values.

Although the system is designed for general-purpose uses, it could provide resource discovery by publishing resource advertisements in the nodes responsible for the multidimensional range that includes the specific values of each resource. Therefore, it would use both pull and push methods, having queries return the advertisements with the addresses of the required resources. However, these gives nodes control over the access to resources in a specific range, with the security risks that this implies.

4.1.22. Mercury. Mercury [Bharambe et al. 2004] is a system intended for the storage of objects defined by multiple attributes and their retrieval through range queries. To attain this objective, Mercury creates a network for each of the existing attributes, called an attribute hub. These hubs are organized similarly to ring DHTs as Chord or Pastry. However, object identifiers are not cryptographically hashed, but they use their values for the corresponding attribute. Thus, they maintain their logical order, and range queries only need to be routed to the node nearest to the beginning of the range and start a sequential search from there.

The value space for each attribute is divided into ranges. As the population of the range is not expected to be distributed uniformly, load balancing techniques must be used. Basically, Mercury subdivides highly loaded ranges, creating a denser population of nodes where there is a denser population of objects. In order to enhance routing, they have links to nodes at certain distances as well as links to successors and predecessors, much like DHTs. These links are set through a random sampling process. Nodes also have links to other attribute hubs, in order to route queries to the appropriate subnetwork. All these links are replicated for fault tolerance. The authors present results that show Mercury's scalability with up to 50000 nodes, where routing takes less than 15 hops.

In order to use Mercury for resource discovery, it has to resort to the use of advertisements which describe resources. These are stored in a node in each attribute hub, assigned to a range of values covering the value of the resource for such attribute. In order to issue a range query, the issuer must determine the attribute for which the query is most selective to minimize the number of nodes that must be contacted.

The unbalanced load distribution can be leveraged by attackers to gain control of the system. Since nodes can decide where to join, a malicious node could join to a desired, highly populated range and gain control of the desired range of attribute values, effectively forbidding access to existing resources in that range and with the possibility to advertise malicious resources instead. The load balancing techniques require the highly loaded node to make a request for a lightly loaded node to disconnect and reconnect in the suggested position. A malicious node could refuse all such requests issued by other nodes, and even inject requests in order to move other nodes to its convenience.

4.1.23. Prefix Hash Tree. Prefix Hash Trees [Ramabhadran et al. 2004] or PHTs are conceived to build an additional layer on top of generic DHT functionalities that allow to perform range searches over a single value. Each object has assigned a value that can be viewed as a string of a determined length. These objects are organized according to prefixes in a trie. The function of PHT is to distribute such trie over a DHT.

Each prefix is hashed and assigned to the corresponding node in the DHT. The structure of the trie is created on-the-fly, with it being expanded and contracted as the number of objects stored in a leaf node increase or decrease past an established threshold. Since the DHT is used to locate each node of the trie, additional state is not necessary, enhancing fault tolerance. However, this implies that trie traversal requires a number of DHT lookups, each with its (usually logarithmic) own cost.

Simulations with a DHT of 1000 nodes test the suboptimality of range queries, that is, how many leaf nodes more than the optimal are traversed in range queries, as well as the load balance of the system. The former show that the number of nodes traversed is around 1.4 times the optimal. The latter shows that about 80% of the nodes receive less than 0.4.

PHTs can be used to build a single attribute-based resource discovery system by storing resource descriptions as objects. As in other systems, this means that the resources would need to republish their advertisements periodically to attain fault tolerance and that completeness of results is not guaranteed in dynamic environments, since advertisements can be lost due to node failure for the period until they are republished.

Security is not commented throughout the paper. Since trie nodes can be accessed individually using the DHT, and the trie changes depending on the load, it would be difficult for a malicious node to gain control over a large portion of the trie. It could, however, if it enters the system as a leaf node and it has the ability to prevent the expansion of the trie. However, whether it does or not is not specified in the paper.

4.1.24. FaSReD. FaSReD. [Smith et al. 2008] is intended to be a fast and scalable resource discovery system. It uses a Prefix Hash Tree (PHT), which is built over Chord. Each resource assigns itself a key from the values of its attributes. For example, a certain value of CPU is assigned some bits (010) a value of RAM is assigned another set of bits (100), and the OS a last set of bits (011). The bits corresponding to all the attributes are concatenated to form the id of the node (010100011). Resources are inserted into the tree in the position that corresponds to its id. Queries can find resources by searching the DHT for a specific id, generated itself from the specified requirements

FaSReD supports range queries by using the maximum common prefix among its maximum and minimum id values to identify a point to start the search. FaSReD uses the DHT to efficiently find the node which acts as root of the subtree of interest. From this point, a PHT search is performed to find those resources that meet the requirements. The main problem with this method is that, since attributes are ordered in the ids, queries with a wide range in its first attributes will start its search in a high level and will have to visit a high number of nodes. However, with an intelligent distribution of attributes, the root of the tree can be freed from most of the responsibility and load that it would have in an ordinary PHT. Nevertheless, it requires that the number of attributes and ranges is fixed beforehand, hampering its flexibility, as well as knowing the distribution of attributes values that will be queried in order to optimize their performance.

In order to have a balanced tree, FaSReD requires all values to be uniformly distributed among the range of the attribute. The mechanism is complete in a stable environment, but failures have not been considered in the maintenance of the tree. The tests presented in the paper show an acceptable scalability and performance, although the overhead that would result from failures is not considered. Security is also not considered throughout the paper, and therefore, nodes responsible for ranges can deny access to resources in their range.

4.1.25. Brunner et al. [2012]. Brunner et al. [2012] present a summarization-based mechanism for resource discovery in grids. A grid is composed of a number of domains, each with a central broker which stores a summary of the resources inside the domain. These summary is created using the Cobweb algorithm [Fisher 1987], and it says the probability of a node with certain characteristics existing inside the domain. These characteristics are expressed as range of values for a number of predefined variables.

Brokers in each domain are connected through a Scribe multicast overlay network, used to propagate summaries. This way, all brokers have a summary of each of the domains. When a query which cannot be satisfied in the local domain is sent to a broker, it checks the summaries of each domain to find the one with a highest probability of answering the query. The broker then forwards the query to the selected domain. This broker will try to find resources matching the query in its domain, and forward it to another domain if a matching is not found.

The mechanism is scalable for resource discovery, since having all the summaries in each broker allows finding the fitting domain in 1 message in the average case. However, the transmission costs of the summaries may be high in large networks. This can be reduced by creating less accurate summaries, which have a lower size but may give place to more costly discovery, since queries may be incorrectly forwarded to a domain that cannot satisfy them.

Fault tolerance is provided by the use of Scribe on top of a Pastry overlay network, although failures in brokers would require retransmitting all the summary information, making recovery costly. Security is not discussed in the paper, but since each broker receives information directly from each domain, even malicious brokers could only send false information about their domain and have limited impact in the overall performance of the system (the effects of a malicious broker claiming to be fitting for all queries could be countered, e.g., by blacklisting).

4.1.26. Yan et al. [2009]. The paper by Yan et al. [2009] presents a resource discovery mechanism built on top of a publish/subscribe mechanism. It can be used for both static and dynamic resources, and supports one-time discovery (i.e., the default behavior of the other presented mechanism, where a client explicitly issues a query when it requires resources) as well as continuous discovery. The latter is intended for clients which have continuing interest in resources fulfilling a set of requirements.

The resource discovery mechanism uses publish/subscribe to send information. While no explicit dependency on a specific underlying mechanism is mentioned in the paper, the mechanism seems to be based on PADRES [Fidler et al. 2005], which also has been used for the evaluation.

The system is composed of a network of interconnected brokers (apparently forming an unstructured network, although it is not explicitly mentioned). Brokers form a tree when needed to distribute information among them and to clients.

In one-time discovery, resource ads are sent to all the brokers. This way, when a client wants to find resources, it only needs to send a query (a subscription in publish/subscribe terms) to the broker to which it is connected. In the case of static resources, the broker stores the received ads, so it can tell the client where to find the resources. In the case of dynamic resources which can change their characteristics over time, the broker stores an initial ad sent by each resource containing the possible range of values it can adopt. The resource sends ads to its broker when its characteristics change, but the broker does not broadcast this ad. Therefore, when a query is forwarded to the broker, it can decide whether the resource currently satisfies it or not.

Continuous discovery mechanisms have brokers store subscriptions as well as publications. For static resources, brokers store the subscriptions of its connected nodes. When receiving a matching publication, they forward it to the subscriber. In the case of dynamic resources, the subscription is forwarded to the brokers which could potentially answer it, just like with one-time queries. However, in this case the broker stores the subscription. When a dynamic resource changes its state, its broker checks the subscriptions for the resource and informs the subscribers interested in the current resource's state.

These mechanisms are not scalable since all brokers must store information of each of the resources present in the system. Some improvements can be obtained through *similarity forwarding*. This involves detecting subscriptions which are *covered* by others, that is, all publications satisfying one also satisfy the other. By not storing covered subscriptions, some storage and communication can be saved in the brokers.

The mechanism can tolerate node churn by adding an expiration date to ads and subscriptions. However, mechanisms for tolerance of broker failures are not presented. Security is not mentioned either. Without additional measures, a malicious broker could prevent the dissemination of ads and queries.

4.2. Unstructured

4.2.1. Iamnitchi et al. [2002]. The papers by Iamnitchi et al. [Iamnitchi and Foster 2001; Iamnitchi et al. 2002] present some approaches for peer-to-peer resource discovery in grid environments. Their system model is composed of a number of organizations which want to share resources among them. Each of these organizations has a local, centralized or hierarchical resource discovery system which can be used to find resources satisfying a certain set of requirements. The problem that they address is how to efficiently find resources in a large-scale network formed by many organizations.

They propose that each organization or site has a dedicated server acting as a supernode, which would have information about all the resources in its site. Supernodes of different sites are connected in a peer-to-peer fashion, forming an unstructured overlay network. Queries that cannot be answered locally by the site's resource discovery system are then forwarded through the peer-to-peer network to other sites. When the required resources are found, the node that can provide the resources replies directly the node that issued the query, which in turn forwards the answer to the user.

They show four different techniques for query forwarding on the defined overlay network. The first one is random, where the node forwards the query to a neighbor chosen randomly among the list of neighbors stored by the node. The second one is experience based and random, where a node forwards a query to a node that answered similar requests previously. This requires the collection of information as the system operates and mechanisms to detect query similarity. In case no relevant information is available, the node just forwards the query to a random neighbor. The third approach is best-neighbor, where the node records the number of responses received from each neighbor. Queries are forwarded to the node that answered more requests, without considering similarity of the queries answered. Finally, experience based and best-neighbor uses the experience-base approach to choose a neighbor to forward the query. When no relevant information is present, the best-neighbor policy is used instead.

In the experiments presented the experience based and random approach happens to obtain the best results. Best-neighbor approach is good for scenarios where the resource distributions are unbalanced, and a small number of nodes offer most of the available resources. This makes plausible the assumption that nodes that returned a certain type of resources might also provide resources of a different kind. However, when the load is balanced and all nodes provide resources in a similar quantity, best-neighbor performs worse than the random approach.

Since queries are processed locally by supernodes which only need to manage a supposedly small number of clients and resources, the search flexibility of the system could be high. However, for the experience-based policy, similarity among queries needs to be determined, which will likely place constraints on the flexibility of queries.

Intra-organization configuration is static, so failures are repaired manually by organization managers or by fault-tolerance mechanisms established internally. However, the failures of any number of nodes will not isolate other nodes, because the overlay is dynamically maintained when responses to queries are received by previously unknown nodes.

With the experience-based forwarding policy, queries can find resources in large networks with a small number of hops, achieving a good scalability. Moreover, information about previous responses, in conjunction with grid security and authentication mechanisms, can serve as a kind of trust mechanism to avoid attacks, letting malicious nodes little capacity of action. However, the mechanism is not complete and when resources of a certain type are scarce, the performance of queries can drop notably.

4.2.2. Mastroianni et al. [2005]. The paper by Mastroianni et al. [2005] presents a grid where each site has a supernode that is connected to other supernodes forming a peer-to-peer network. Nodes in a site send their queries for resources to the local supernode. This checks the local information service that operates inside the site, like Globus' MDS-2 [The Globus Alliance 2013]. If not enough resources that satisfy the query are found, the supernode uses the overlay peer-to-peer network to forward the query to other sites.

In order to find resources in other sites, the supernode sends the query to a subset of its neighbors. This subset is empirically determined to contain the neighbors that provided a higher number of responses in previous queries. However, this approach was shown to be the worst performing of all the analyzed in the papers by Iamnitchi et al. [Iamnitchi and Foster 2001; Iamnitchi et al. 2002].

The query language is not specified, and the paper only mentions that it should be a standard language that supernodes of all sites can understand. Moreover, this language should be translatable to that of the local information service of each site. Therefore, the flexibility of the query language will depend on the systems deployed at each grid site, although potentially it could have a great flexibility since queries are processed in centralized or hierarchical networks of small size.

The system relies on infrastructure servers at each site, so fault tolerance should be locally provided. However, no mention is done of simple policies like replication of supernodes. Neighbors in the overlay networks are only chosen when a supernode joins the grid, so there is the possibility that failures and disconnections end up leaving a nonfailed node without active neighbors. Security is also provided by the grid mechanisms working at each site. However, even in the case of malicious nodes present in the overlay network, queries should obtain correct responses since they are forwarded along multiple paths. Grid security mechanisms should prevent the use of faulty resources pointed by a malicious node that receives a query.

The scalability of the system with respect to information maintenance is high, since nodes only need to know about a constant number of neighbors. Regarding communication cost of queries, in cases where there is a large number of resources that satisfy a query, there is a high probability that responses will be received in a small number of hops between sites. However, if the required type of resources is scarce, the search will require many hops until a result can be found, since the search is not directed, and the best-neighbor algorithm does not provide the best results in this case.

4.2.3. Cluster Computing On-the-Fly. Cluster Computing On-the-Fly (CCOF) [Lo et al. 2004] is designed to harvest idle cycles from computers connected to the Internet. In a paper [Zhou and Lo 2004] the authors focus on the resource discovery component of CCOF and define a general framework for it. In their model, nodes form an unstructured overlay network which can be used to forward queries. Once a query arrives at a node that can satisfy it, the node sends a response to the one who issued the query. Once it has received enough answers, it proceeds to contact the nodes that will execute the required application.

The authors evaluate four search methods. The first of these methods is expanding ring search, where a node sends a request to its direct neighbors and they send a response if they can execute the task. If not enough responses are received, then it sends the query to peers one hop farther. This process is repeated until enough responses are found or the search reaches the scope limit. The second method is random walk search, where the node forwards the query to k random neighbors. These neighbors answer to the node if they can complete the task, and forward it to k random neighbors until the scope limit is reached. The third method is ads-based search, where each node forwards its profile information, that is, its resource specification, to its neighbors. They cache and forward it until its scope limit is reached. When a node needs to execute a job, it looks in its list of cached ads for a node that can respond to the request. Finally, the rendezvous point search assumes that there are a set of rendezvous points that nodes can access to locate available hosts. In order to do this, nodes send their advertisements to the nearest rendezvous point. It is assumed that rendezvous points are placed geographically scattered and in sufficient number for the network to perform correctly by an out-of-band mechanism. Nodes discover rendezvous points by asking their neighbors. These four approaches are compared to a centralized resource discovery mechanism that serves as an optimum reference.

The scalability of the system is tested with 4000 nodes, and the rendezvous points mechanism is shown to be the best in terms of number of messages sent, while the ads-based and expanding ring require less hops per query. Fault tolerance is high with respect to the resource discovery process for all the presented algorithms, although the job completion rate of the whole system drops notably with high peer disconnection probabilities. The security of the rendezvous points approach is unknown since it would depend on the absence of malicious nodes among the set of rendezvous points, which are assumed to be placed by an out-of-band method. If this method is not secure, malicious rendezvous points could make their zone of influence nonfunctional. The other approaches seem more secure, since a single node cannot hamper query or ad routing notably.

The presented mechanisms do not achieve completeness, at least in a realistic setting. A node could contact all the rendezvous points in the network in the corresponding approach, but this could be nonpractical depending on the network size and proportion of rendezvous points. The other approaches would require flooding to contact all of the nodes in the network in order to achieve completeness. Accuracy of the expanding ring and random walk mechanisms is high, since the requests are answered directly by the nodes involved. Ads-based and rendezvous points, however, are sensible to stale data, potentially returning nodes that are not available anymore or that have changed some of their dynamic characteristics, like load.

Potentially, all the methods have high query flexibility as the matching is done locally in the candidate node, so any kind of checking could be performed, or over a limited amount of ads. However, the query format explained in the paper only includes number of processors needed and time spent on each processor.

4.2.4. Moreno-Vozmediano [2006]. The paper by Moreno-Vozmediano [2006] presents a resource discovery mechanism for ad hoc grids, composed of wireless devices and forming Mobile Ad hoc NETworks or MANETs. The model presented in the article is a hybrid mechanism, where each node sends advertisements to nodes at distance at most R hops and store the ads they receive. When a node is searching resources, if they are not found locally in the stored ads, a query is sent to nodes at distance R. These nodes, upon receiving the query, look for resources in their local database. If they are found, the node that issued the query is informed. If not, the query can be sent to nodes at distance 2R.

The nodes form a decentralized unstructured overlay network, where queries are processed locally over a small number of resource advertisements, and therefore there is a potentially high query flexibility. Scalability is achieved by limiting the scope of both ad and query forwarding. This also implies that the mechanism is not complete, and only nearby resources can be efficiently found, although its accuracy can be high, only having the risk of stale data if the update periodicity of ads is too long. The system's fault tolerance, however, is not specified. Its security, on the other hand, depends on the specific topology of each network. Malicious nodes can make a part of the network unreachable for their neighbors if the network is not highly connected and alternate paths to each node are not present. However, in general, they cannot have a high impact on the overall network.

4.2.5. Padmanabhan et al. [2010]. Padmanabhan et al. [2010] present a self-organized grouping framework for efficient resource discovery. Their system is divided in groups and composed of two types of nodes: leaders and workers. Each group has one leader, and each leader knows the workers in its group and all the other leaders. Workers know all leaders. Therefore, the network forms a graph of diameter 2. The nodes are also connected through an unstructured overlay network they use for gossiping information about the leaders.

Nodes are defined by a number of attributes which logically place them in a d-dimensional space. This space is divided in hypercubes using space-filling curves. The nodes inside each of these hypercubes form a group. Therefore, a group contains nodes with similar characteristics.

A query for resources is sent to the leader closer to what is required. This leader will search among its workers to find those which can fulfill the query. If no fitting resources are found, the leader will forward the query to another leader which could potentially answer it. If no candidate leader exists, one is randomly selected for forwarding.

There is a trade-off in the number of attributes used for grouping and the efficiency of resource discovery. If many attributes are used, many groups will be formed, which may be too small. In this case, visiting many groups would be required, making resource discovery inefficient. On the other hand, if a client wants to find resources based on an attribute that was not used for grouping, self-organization does not provide any advantage and the search will be random.

Table I. Level of Achievement of Each Requirement, Scored from 1 to 5 Stars for Search Flexibility, Scalability, Fault Tolerance and Security, and with a Yes/No for Completeness and Accuracy, for Each System

	Search flexibility	Scalability	Fault-tolerance	Completeness	Accuracy	Security
Ranjan et al. 07	★★★	★★★	★★★★	No	Yes	★★★★
Chang-yen et al. 08	★★★	★★★★	★★★★	No	Yes	★★★
Xenosearch	★★★	★★★	★	Yes	No	★★★
Abdullah et al. 09	★★★★	★★★★	★	No	Yes	★★★★
Abdullah et al. 11	★★★★	★★★★	★★★★	No	Yes	★★
SWORD (SingleQuery)	★★★★★	★★★	★★★★	Yes	Yes	★★★★
Cheema et al. 05	★★	★★★	★★★★	Yes	Yes	★★
Squid	★★★	★★★	★★★★	Yes	Yes	★★
DGRID	★	★★★★	★★★★	Yes	Yes	★★★★★
MAAN	★★★	★★★	★★★★	Yes	Yes	★★★
Self-Chord	★	★★★★	★★★★	Yes	Yes	★
Kim et al. 07	★★	★★★	★★★	No	Yes	★★★
Costa et al. 09	★★★	★★★★	★★★★★	Yes	Yes	★★★★
MURK	★★★	★★★	★★★	Yes	Yes	★★★
NodeWiz	★★★	★★★	★★	Yes	Yes	★★
NR-Tree	★★★	★★★★	★★★★	Yes	Yes	★★★
Cone	★	★★★	★★★★	Yes	Yes	★★★
Gao et al. 04	★★★★	★★★	★★★★	Yes	Yes	★★★
Tanin et al. 07	★★★	★★★	★★★	Yes	Yes	★★
P-tree	★	★★★★	★★★★	Yes	Yes	★★★★
SCRAP	★★★	★★★	★★★★	Yes	No	★★★
Mercury	★★★	★★★	★★★★	Yes	Yes	★★
PHT	★	★★★	★★★★	Yes	Yes	★★★
FaSReD	★★★	★★★	★	Yes	No	★★
Brunner et al. 11	★★★	★★★★	★★★	No	No	★★
Yan et al. 09	★★★★	★★	★	Yes	Yes	★★
Iamnitchi et al. 01 (experience-based)	★★★★	★★★★★	★★★★★	No	Yes	★★★★★
Mastroianni et al. 05	★★★★	★★★★★	★★★★	No	Yes	★★★★★
CCOF (Rendezvous Points)	★★★★	★★★★★	★★★	No	Yes	★★★
Moreno-Vozmediano 06	★★★★	★★★★★	★	No	Yes	★★★★
Padmanabhan et al. 10	★★★	★★★★	★★★★	No	Yes	★★★

Scalability of this mechanism depends mainly on the number of groups that are created, since each node must know all leaders. Therefore, if the number of groups is too large, a large amount of information must be maintained by the nodes. However, the maintenance of this information is efficient and fault tolerant thanks to the use of a gossiping substrate. Security is not mentioned in the paper, but apparently malicious leaders could only deny access to their groups. If groups are not too large, this means making a small part of the network unavailable.

5. SUMMARY AND DISCUSSION

In this section we compare the characteristics of each of the surveyed systems. In Table I we present the degree of achievement of each of the requirements defined at the beginning of the article, for each of the presented systems. These are search flexibility, scalability, fault tolerance, and security, which are scored in a continuous scale from 1 to 5; and completeness and accuracy, which are classified as a binary value. In general terms, the higher the number, the better achievement of the requirement by the system. Next we show an approximate description of the meaning of some values for each of the requirements, in order to help the understanding of the characteristics of each system. Note that these descriptions might not accurately represent the value assigned to some of the systems.

A qualification of 1 in search flexibility means that the system only uses a single value to classify resources, allowing for queries requesting values equal, greater, or lesser than a given value. A qualification of 2 means that the system allows multiple, predefined attributes to be used to define resources, and queries can require exact values for these attributes, while 3 means that values within a given range can also be requested. A qualification of 4 means that range queries can be issued over a variable, nonpredetermined number of attributes, and 5 adds the possibility to request inter-node properties, like finding nodes that are nearby.

We measure scalability from a theoretical point of view, considering the asymptotic cost of increasing scale. Experimental results regarding scalability of each system will be presented later, as shown on the respective papers. Here we consider two factors that affect scalability: the cost of information maintenance at each node, and the cost of issuing queries. We define the scalability of a system as a function of how these two costs increase with scale. A qualification of 1 means that both costs increase linearly with network size or worse. A qualification of 2 is for systems where one of those costs grows logarithmically while the other grows linearly, and a 3 is for those where both costs grow logarithmically. A 4 is given to systems where one of the two costs grows logarithmically with network size while the other is constant, and finally a 5 means that both costs are constant.

Regarding fault tolerance, we qualify with 1 those systems where fault tolerance is not considered at all, no recovery mechanisms are presented, and failures or disconnections can destroy the connectivity of the overlay network with no given way of reconfiguration. In many of the systems that are given a 1, fault tolerance is stated to be a subject of future work, although some specifically require stable servers to work. A 2 means that the system can deal with node disconnections, but not with unpredicted failures, and that the reconfiguration can have a high cost. A qualification of 3 means that the system can deal with both ordered disconnections as well as crash failures, although it requires an important reorganization that might have a high overhead. A qualification of 4 means that the system can deal with churn and crash failures with only a moderate cost, while 5 means that almost no reconfiguration is required in case of failure.

Since security analysis of each of the systems presented is not given in the literature, we only make here a basic classification based on the influence that malicious nodes could have in the working of each of the resource discovery mechanisms. Note that these classifications are based on some assumptions previously presented when discussing each system. We classify with a 1 systems where the presence of one or a few malicious nodes could turn all the system nonfunctional. A 2 is given if a great part of the system could become nonfunctional or unavailable to the rest of the system by the action of one or a few malicious nodes. A qualification of 3 denotes a system where a small number of malicious nodes can turn a small part of the system nonfunctional, or a great part of the system unavailable for one or a set of legitimate nodes. A 4 is appropriate where malicious nodes could make a small part of the system unavailable for one or a set of nodes, while 5 means that malicious nodes cannot make any part of the system nonfunctional or unavailable to any node.

Table II shows the specific behavior of each system in resource description, query value selectivity, and theoretical lookup complexity, where available. The former two are classified according to the taxonomies presented earlier, while the latter is shown

Table II. Resource Description Supported and Query Value Selectivity for Each System, and Theoretical Lookup Complexity where available

System	Number of attributes	Query selectivity	Lookup complexity
Ranjan et al. 07	Multiple	Range	$\theta(E[T] \times (log\,N + f_{max} - f_{min}))$; T is the number of disjoint paths, $f_{min,max}$ are the minimum and maximum levels of division of the space
Chang-yen et al. 08	Multiple	Range	N.A. (uses multiple Pastry networks)
XenoSearch	Multiple	Range	N.A. (uses multiple Pastry networks)
Abdullah et al. 09	Variable (auction)	Any (auction)	$O(log_{2^b}\,N)$; b is 4 by default
Abdullah et al. 11	Variable (auction)	Any (auction)	$O(N_{mm})$; N_{mm} is the number of matchmakers
SWORD (SingleQuery)	Variable	Range	$O(m + log\,N)$; m is the number of peers in the selected range
Cheema et al. 05	Multiple	Exact match	$O(log\,N)$
Squid	Multiple	Prefix	$n_c \times O(log\,N)$; n_c is the total number of isolated clusters in the index space
DGRID	One	Exact match	$O(min(log\,K, log\,N))$; K is the number of resource types
MAAN	Multiple	Range	$O(log\,N + N \times s_{min})$; s_{min} is the minimum selectivity for all attributes
Self-Chord	One	Range	N.A. (99th percentile lower than $log\,N$)
Kim et al. 07	Multiple	Exact match and inequality	N.A. (uses CAN)
Costa et al. 09	Multiple	Range	N.A.
MURK	Multiple	Range	N.A.
NodeWiz	Multiple	Range	N.A. (uses a binary tree)
NR-Tree	Multiple	Range, k-nearest neighbor	N.A. (uses CAN and R*-tree)
Cone	One	Inequality	$O(log\,N)$
Gao et al. 04	Variable	Range	$O(log\,R_q)$; R_q is the query's length
Tanin et al. 07	Multiple	Range	$O(log\,N + f_{max} - f_{min})$; $f_{min,max}$ are the minimum and maximum levels of division of the space
P-tree	One	Range	$O(m + log_d\,N)$; d is the order of the tree, m the number of results
SCRAP	Multiple	Range	N.A. ($O(log\,N)$ for a single 1-d range)
Mercury	Multiple	Range	$O(1/k \times log^2\,N)$; k is the number of long distance links
PHT	One	Prefix	$O(log\,D \times log\,N)$; D is the number of bits of ids
FaSReD	Multiple	Range	$O(log(log\,N) \times log\,N)$; close to optimal: $O(M)$, where M is the number of attributes
Brunner et al. 11	Multiple	Range	$O(1)$
Yan et al. 09	Multiple	Range	$O(1)$ for the static case; for the dynamic case, it depends on the publish/subscribe implementation
Iamnitchi et al. 01 (experience-based)	Variable	Range	$O(1)$, bounded by a TTL ($O(N)$ for complete results)
Mastroianni et al. 05	Variable	Range	$O(1)$, bounded by a TTL ($O(N)$ for complete results)
CCOF (Rendezvous Points)	Variable	Range	N.A.
Moreno-Vozmediano 06	Variable	Range	N.A.
Padmanabhan et al. 10	Multiple	Range	N.A.

as asymptotic complexity. In lookup complexity, measures are given in function of the total number of nodes N. We only show them when they are presented by the authors, except in the case of Squid, where we present the lookup complexity inferred by Ranjan et al. [2008].

In Table III, we show a summary of the design characteristics of each of the presented systems. We specify the topology of the overlay network, mentioning the general classification as well as the specific implementation used when available. We also mention the characteristic of the underlying topology that emerges for systems where it does.

Specifically, most systems group hosts by attribute values, putting together those with similar attributes. The system by Chang-Yen et al. [2008] uses the network proximity metric defined by Pastry. The system by Abdullah et al. [2011] creates resource categories and groups resources in the same category. DGRID uses the grid domains to which nodes belong to create the topology of the overlay network. NodeWiz groups nodes according to query distribution, creating a topology which is more efficient to answer the queries which are more commonly issued.

Figures 3 and 4 show the trade-offs of each system. They both place systems in the space according to their search flexibility (x-axis) and scalability (y-axis). Additionally, the size of the dot that marks the position of each system denotes its fault tolerance, with a more faulttolerant system having a bigger representation.

Figure 3 shows complete systems, while Figure 4 presents incomplete systems. We separated them in these two categories since the understanding of their scalability must take into account the question of completeness. Incomplete systems might be highly scalable in an asymptotic measure, since the number of messages generated by a query can be limited by a constant, for example, a TTL value assigned to the query. However, this means that the possibility of finding results is also limited by this constant. In complete systems, on the other hand, the number of messages needed to respond a query is usually dependent on the network size, and cannot be artificially limited to a constant value since these would eliminate the guarantee of obtaining complete results. Therefore, complete systems might yield a much better performance than incomplete systems that have a better asymptotic lookup complexity, since the latter might only find suitable resources when they are abundant.

Most complete systems have a logarithmic lookup complexity (Figure 3). The most adopted mechanism is to have resources publish themselves to a certain location, and queries sent to that same location to retrieve the resources' data. DHTs' logarithmic complexity is seen as the target cost for both processes. Only four systems have a better scalability by reducing the cost of resource publication to constant.

Complete systems exhibit a high variety of search flexibility, ranging from systems that only allow one attribute to define resources to others which allow any combination of <key, value> pairs. SWORD is the most flexible system surveyed according to its description in Oppenheimer et al. [2005]. However, the implementation deployed on PlanetLab and available to any user (SWORD) does not offer support for inter-node requirements, therefore reducing the search flexibility of the system in a practical environment.

In total, half of the surveyed complete systems are located at the same point. These have a logarithmic lookup and publication complexity, and support range searches for a fixed number of attributes. Moreover, most complete systems also depend on DHTs for their fault tolerance, achieving a fairly good value, with some exceptions. The positive exception is the system by Costa et al. [2009], which uses a random gossiping protocol to attain a very high fault tolerance.

Half of the surveyed incomplete systems have equivalent scalability and flexibility (Figure 4). They use unstructured networks where the number of messages generated per query is limited by a TTL, providing a constant cost for searches. Their flexibility is

Table III. Summary of Some of the Design Characteristics of Each System

System	Topology	Emergent underlying topology	Local centralization	Total centralization	Information spread method
Ranjan et al. 07	Ring DHT (Chord)	No	Yes	Medium	Pull/push
Chang-yen et al. 08	Ring DHT (Pastry)	Yes (proximity metric)	Optional	Very low	Push, pull if it fails
XenoSearch	Ring DHT (Pastry) and aggregation trees	No	Yes	Medium	Pull/push
Abdullah et al. 09	Ring DHT (Pastry)	No	No	Medium-high	Pull/push
Abdullah et al. 11	Ring DHT (Pastry)	Yes (resource category)	No	High	Pull/push
SWORD (SingleQuery)	Ring DHT (Bamboo)	No	No	Low	Pull/push
Cheema et al. 05	Ring DHT (Pastry)	No	No	Very low	Pull/push
Squid	Ring DHT (Chord)	No	No	Very low	Pull/push
DGRID	Ring DHT (Chord)	Yes (domains)	No	Very low	Pull
MAAN	Ring DHT (Chord)	No	No	Very low	Pull/push
Self-Chord	Ring DHT (Chord)	Yes (attribute values)	No	Very low	Pull/push
Kim et al. 07	Multidimensional space DHT (CAN)	No	No	Very low	Pull
Costa et al. 09	Multidimensional space	Yes (attribute values)	No	Very low	Pull
MURK	Multidimensional space	No	No	Low	Pull/push
NodeWiz	Distributed decision tree	Yes (query distribution)	No	High	Pull/push
NR-Tree	Multidimensional space and local R*-trees	Yes (attribute values)	Optional	Medium	Pull/push
Cone	Trie and ring DHT	Yes (attribute values)	Optional	Medium-low	Pull
Gao et al. 04	Ring DHT (Chord) and Range Search Tree (RST)	No	No	Very low	Pull/push

Tanin et al. 07	Ring DHT (Chord) and quadtree	No	No	Low	Pull/push
P-tree	Ring DHT (Chord) and partial B+ trees	Yes (attribute values)	No	Very low	Pull
SCRAP	Skip graph	No	No	Low	Pull/push
Mercury	Multiple ring networks	Yes (attribute values)	No	Very low	Pull/push
PHT	Trie over a DHT	No	No	Very low	Pull/push
FaSReD	Ring DHT (Chord) and tree (PHT)	Yes (attribute values)	No	Medium	Pull/push
Brunner et al. 11	Multicast tree (Scribe) over a ring DHT (Pastry)	No	Yes	High	Pull/push
Yan et al. 09	Trees and unstructured	No	Yes	Low	Pull/push
Iamnitchi et al. 01 (experience-based)	Unstructured	No	Yes	Low	Pull
Mastroianni et al. 05	Unstructured	No	Yes	Medium-low	Pull
CCOF (Rendezvous Points)	Unstructured	No	No	Medium-low	Pull/push
Moreno-Vozmediano 06	Unstructured	No	No	Very low	Pull/push
Padmanabhan et al. 10	Unstructured	Yes (attribute values)	No	Low	Pull

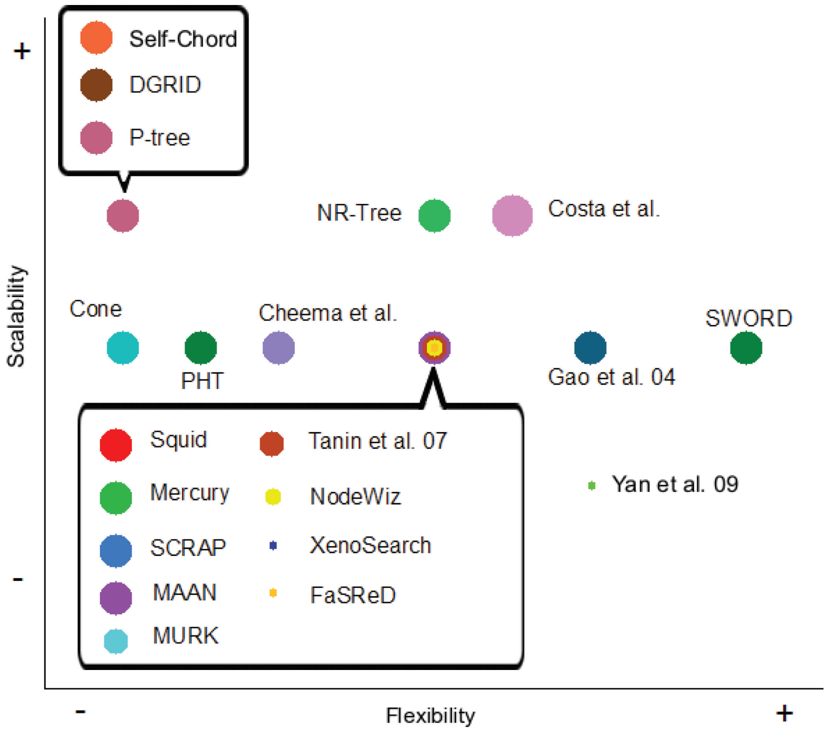

Fig. 3. Complete systems organized for their search flexibility and scalability, with size meaning fault tolerance. Systems with same flexibility and scalability are listed separately for clarity. Most of them have a logarithmic lookup cost.

high since the matching between queries and resource descriptions is evaluated locally. Therefore, it can include a variable number of attributes. The other systems make use of a variety of mechanisms over structured networks.

6. CONCLUSIONS

The relation among the different requirements faced by resource discovery systems and the design approaches taken by different proposals can be seen through the comparison of the systems discussed in this survey.

The first thing that we see is that having a query language with enough flexibility to specify range values for a set of attributes is required for any system that aims at deployment in a real environment. However, systems that use DHTs and similar methods to organize information in a distributed index usually require beforehand knowledge of the defined attributes. Moreover, they might also require knowledge of the distribution of values that will be present and queried, either in order to optimize the organization of the index for better performance or in order to work at all. Query-forwarding techniques for unstructured topologies, on the other hand, require each node that processes the query to locally compare it with the data stored by it about available resources, which allows for a greater flexibility.

Only one of the discussed systems, SWORD, presents support for inter-node properties. These properties are evaluated by a centralized optimizer, executed locally by the node issuing a query over the returned results. Therefore, many other systems could make use of such an optimizer to further refine the results of their queries, incorporating inter-node relations or other types of properties. The information

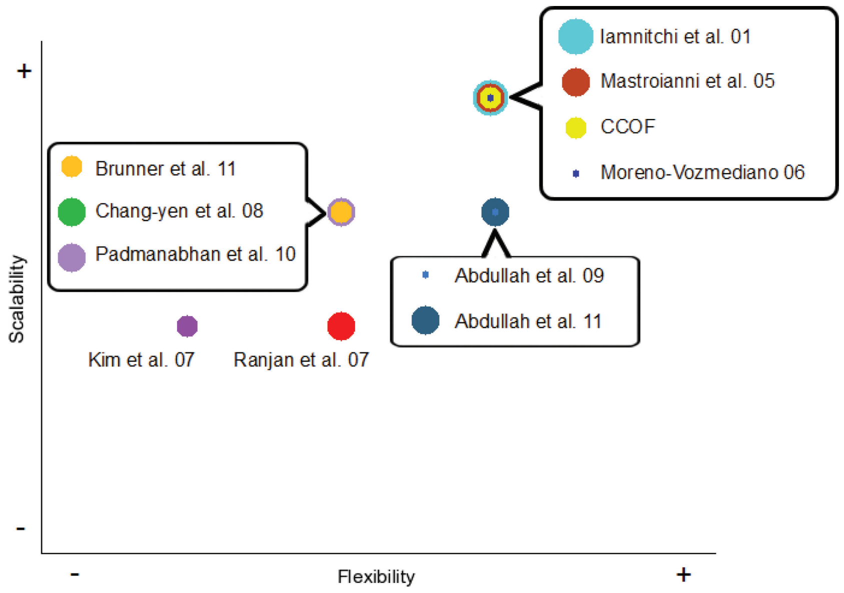

Fig. 4. Incomplete systems organized for their search flexibility and scalability, with size meaning fault tolerance. Systems with same flexibility and scalability are listed separately for clarity. Half of the systems have high flexibility and cost bounded by a constant.

required to perform this evaluation is obtained through a mechanism of representatives, which are a subset of the nodes of the networks. The latency among the representatives is measured and stored, and the latency between any two nodes is considered to be that of their assigned representatives. This mechanism is independent from the one used to obtain resources that satisfy individual requirements. However, the implementation of SWORD deployed in PlanetLab does not provide support for these type of queries. We believe that more research on methods to efficiently provide such evaluation of inter-node requirements is needed to enable their use in real dynamic systems.

Unstructured topologies offer a better scalability in general, since resources can be found with a constant cost, derived from fixed values like the TTL of queries. However, this is only true when required resources are frequent, that is, there is a great amount of resources available to satisfy any given query. In order to find rare resources, completeness is required. This is provided by organized indexes based on structured topologies, but at a cost of greater overhead. This overhead comes in the form of topology maintenance and reconfiguration messages, periodical advertisement messages in most systems, which use a combination of pull and push techniques, or longer paths toward resources, among others. Moreover, some systems that use structured topologies provide complete answers to range queries at a great cost, where in most cases completeness is not required. However, this overhead can be easily surpassed by the cost of finding rare resources in unstructured systems, which might require flooding the whole network with a query. These are clear trade-offs that must guide the selection of a specific resource discovery mechanism for a system depending on the environment where it will be deployed and the requirements that it has to meet.

One aspect that many systems do no develop enough is fault tolerance. Many systems rely on the fault tolerance claimed by the overlay network, without considering the specific aspects of the resource discovery mechanisms that might hamper performance in presence of failures, or even render the system unusable. Similarly, security is seldom considered, and while most systems cannot be victims of denial-of-service attacks in their totality, they are vulnerable to attacks rendering the system partially nonfunctional or unavailable. Moreover, malicious nodes can often control what resources are returned by a significant proportion of queries, requiring additional security guarantees for the remote execution mechanisms.

REFERENCES

Abdullah, T. and Bertels, K. 2011. Nature-inspired self-organization in p2p ad hoc grids. In *Proceedings of the 7th International Conference on Autonomic and Autonomous Systems (ICAS'11)*. 134–139.

Abdullah, T., Mhamdi, L., Pourebrahimi, B., and Bertels, K. 2009a. Resource discovery with dynamic matchmakers in ad hoc grid. In *Proceedings of the 4th International Conference on Systems (ICONS'09)*. IEEE Computer Society, Los Alamitos, CA, 138–144.

Abdullah, T., Alima, L. O., Sokolov, V., Calomme, D., and Bertels, K. 2009b. Hybrid resource discovery mechanism in ad hoc grid using structured overlay. In *Proceedings of the 22nd International Conference on Architecture of Computing Systems.* M. Berekovic, C. Mller-Schloer, C. Hochberger, and S. Wong, Eds., Lecture Notes In Computer Science, vol. 5455, Springer, 108–119.

Aspnes, J. and Shah, G. 2003. Skip graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, Philadelphia, PA, 384–393.

Basu, S., Banerjee, S., Sharma, P., and Lee, S. 2005. NodeWiz: Peer-to-peer resource discovery for grids. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05)*. Vol. 1, IEEE Computer Society, Los Alamitos, CA, 213–220.

Beckmann, N., Kriegel, H., Schneider, R., and Seeger, B. 1990. The r*-tree: An efficient and robust access method for points and rectangles. *SIGMOD Rec. 19*, 2, 322–331.

Bhagwan, R., Mahadevan, P., Varghese, G., Voelker, G. M. 2004. Cone: A distributed heap-based approach to resource selection. Tech. rep. CS2004-0784, University of California, San Diego.

Bharambe, A. R., Agrawal, M., and Seshan, S. 2004. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'04).* ACM Press, New York, NY, 353–366.

Brunner, R., Caminero, A., Rana, O., Freitag, F., and Navarro, L. 2012. Network-aware summarisation for resource discovery in p2p-content networks. *Future Gener. Comput. Syst. 28*, 3, 563–572.

Cai, M., Frank, M., Chen, J., and Szekely, P. 2003. MAAN: A multi-attribute addressable network for grid information services. In *Proceedings of the 4th International Workshop on Grid Computing*. IEEE Computer Society, Los Alamitos, CA, 184.

Chang-Yen, I., Smith, D., and Tzeng, N. 2008. Structured peer-to-peer resource discovery for computational grids. In *Proceedings of the 15th ACM Mardi Gras Conference: From Lightweight MashUps To Lambda Grids: Understanding the Spectrum of Distributed Computing Requirements, Applications,Tools, Infrastructures, Interoperability, and the Incremental Adoption of Key Capabilities (MG'08)*. Vol. 320, ACM Press, New York, NY, 1–8.

Cheema, A. S., Muhammad, M., and Gupta, I. 2005. Peer-to-peer discovery of computational resources for grid applications. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. IEEE Computer Society, Los Alamitos, CA, 179–185.

Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. 1999. Freenet: A distributed anonymous information storage and retrieval system. Freenet white paper. http://freenetproject.org/freenet.pdf.

Costa, P., Napper, J., Pierre, G., and Steen, M. V. 2009. Autonomous resource selection for decentralized utility computing. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS'09).* IEEE Computer Society, Los Alamitos, CA, 561–570.

Cotop. 2013. home page. http://codeen.cs.princeton.edu/cotop/.

Crainiceanu, A., Linga, P., Gehrke, J., and Shanmugasundaram, J. 2004. Querying peer-to-peer networks using p-trees. In *Proceedings of the 7th International Workshop on the Web and Databases (WebDB'04), collocated with ACM SIGMOD/PODS*. Vol. 67, ACM Press, New York, NY, 25–30.

Delmastro, F., Conti, M., and Gregori, E. 2006. P2p common api for structured overlay networks: A cross-layer extension. In *Proceedings of the International Symposium on World of Wireless, Mobile and Multimedia Networks.* IEEE Computer Society, Los Alamitos, CA, 593–597.

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. 2005. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Comm. Surv. Tutorials* 7, 2, 72–93.

Fidler, E., Jacobsen, H., Li, G., and Mankovski, S. 2005. The padres distributed publish/subscribe system. In *Feature Interactions in Telecommunications and Software Systems VIII*, 12–30.

Fisher, D. 1987. Knowledge acquisition via incremental conceptual clustering. *Mach. Learn. 2,* 2, 139–172.

Forestiero, A., Leonardi, E., Mastroianni, C., and Meo, M. 2010. Self-chord: A bio-inspired p2p framework for self-organizing distributed systems. *IEEE/ACM Trans. Netw. 18,* 5, 1651–1664.

Ganesan, P., Yang, B., and Garcia-Molina, H. 2004. One torus to rule them all: Multi-dimensional queries in p2p systems. In *Proceedings of the 7th International Workshop on the Web and Databases (WebDB'04)* colocated with *ACM SIGMOD/PODS.* Vol. 67, ACM Press, New York, 19–24.

Gao, J. and Steenkiste, P. 2004. An adaptive protocol for efficient support of range queries in dht based systems. In *Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04).* IEEE Computer Society, Los Alamitos, CA, 239–250.

Gnutella Protocol Specification Wiki. 2013. http://wiki.limewire.org/index.php?title=GDF.

Iamnitchi, A. and Foster, I. T. 2001. On fully decentralized resource discovery in grid environments. In *Proceedings of the 2nd International Workshop on Grid Computing*. C. A. Lee, Ed., Lecture Notes In Computer Science, vol. 2242, Springer, 51–62.

Iamnitchi, A., Foster, I., and Nurmi, D. C. 2002. A Peer-to-peer approach to resource location in grid environments. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, Los Alamitos, CA, 419.

Kim, J.-S., Nam, B., Marsh, M., Keleher, P., Bhattacharjee, B., Richardson, D., Wellnitz, D., and Sussman, A. 2007. Creating a robust desktop grid using peer-to-peer services. In *Proceedings of the National Science Foundation Next Generation Software Workshop (NSFNGS'07).* IEEE Computer Society, Los Alamitos, CA, 1–7.

Kim, J.-S., Nam, B., Marsh, M., Keleher, P., Bhattacharjee, B., and Sussman, A. 2008. Integrating categorical resource types into a p2p desktop grid system. In *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing.* IEEE Computer Society, Los Alamitos, CA, 284–291.

Kazaa Media Desktop. 2013. http://www.kazaa.com/.

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

Lo, V., Zhou, D., Zappala, D., Liu, Y., and Zhao, S. 2004. Cluster computing on the fly: P2p scheduling of idle cycles in the internet. In *Proceedings of the 3rd International Conference on Peer-to-Peer Systems (IPTPS'04).* Lecture Notes in Computer Science, vol. 3279, Springer, 227–236.

March, V., Teo, Y. M., and Wang, X. 2007. DGRID: A dht-based resource indexing and discovery scheme for computational grids. In *Proceedings of the 5th Australasian Symposium on Grid Computing and e-Research*. Vol. 68, 41–48.

Mastroianni, C., Talia, D., and Verta, O. 2005. A super-peer model for resource discovery services in large-scale grids. *Future Gener. Comput. Syst. 21*, 8, 1235–1248.

Moreno-Vozmediano, R. 2006. Resource discovery in ad-hoc grids. In *Proceedings of the International Workshop on Grid Computing Security and Resource Management (ICCS'06).* Lecture Notes in Computer Science, vol. 3994, Springer, 1031–1038.

Oppenheimer, D., Albrecht, J., Patterson, D., and Vahdat, A. 2005. Design and implementation tradeoffs for wide-area resource discovery. In *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05).* IEEE Computer Society, Los Alamitos, CA, 113–124.

Padmanabhan, A., Ghosh, S., Wang, S. 2010. A self-organized grouping (sog) framework for efficient grid resource discovery. *J. Grid Comput. 8,* 3, 365–389.

Peterson, L., Bavier, A., Fiuczynski, M. E., and Muir, S. 2006. Experiences building planet lab. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation.* USENIX Association, Berkeley, CA, 351–366.

Planetlab. 2013. home page. http://www.planet-lab.org/.

Ramabhadran, S., Ratnasamy, S., Hellerstein, J., Shenker, S. 2004. Prefix hash tree an indexing data structure over distributed hash tables. IRB Tech. rep. http://berkeley.intel-research.net/sylvia/pht.pdf.

Ranjan, R., Chan, L., Harwood, A., Karunasekera, S., and Buyya, R. 2007. Decentralised resource discovery service for large scale federated grids. In *Proceedings of the 3rd IEEE International Conference on E-Science and Grid Computing (E-SCIENCE'07).* IEEE Computer Society, Los Alamitos, CA, 379–387.

Ranjan, R., Harwood, A., Buyya, R. 2008. Peer-to-peer-based resource discovery in global grids: A tutorial. *IEEE Comm. Surv. Tutorials 10,* 2, 6–33.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Schenker, S. 2001. A scalable content-addressable network. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01).* ACM Press, New York, 161–172.

Rowstron, A. I. and Druschel, P. 2001. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*. R. Guerraoui, Ed., Lecture Notes In Computer Science, vol. 2218, Springer, 329–350.

Schmidt, C. and Parashar, M. 2004. Enabling flexible queries with guarantees in p2p Systems. *IEEE Internet Comput. 8*, 3, 19–26.

Schmidt, C. and Parashar, M. 2008. Squid: Enabling search in dht-based systems. *J. Parallel Distrib. Comput. 68*, 7, 962–975.

Sha-1 Standard. 2013. http://csrc.nist.gov/publications/fips/fips180-3/fips180-3 final.pdf.

Smith, D., Tzeng, N., and Ghantous, M. M. 2008. FaSReD: Fast and scalable resource discovery in support of multiple resource range requirements for computational grids. In *Proceedings of the 7th IEEE International Symposium on Network Computing and Applications (NCA'08).* IEEE Computer Society, Los Alamitos, CA, 45–51.

Spence, D. and Harris, T. 2003. XenoSearch: Distributed resource discovery in the xenoserver open platform. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing.* IEEE Computer Society, Los Alamitos, CA, 216–225.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. 2003. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw. 11,* 1, 17–32.

Suryanarayana, G. and Taylor, R. N. 2004. A survey of trust management and resource discovery technologies in peer-to-peer applications. Tech. rep. UCI-ISR-04-6, UCI Institute for Software Research.

Sword. 2013. home page. http://sword.cs.williams.edu/.

Tanin, E., Harwood, A., and Samet, H. 2007. Using a distributed quadtree index in peer-to-peer networks. *The VLDB J. 16,* 2, 165–178.

Thain, D., Tannenbaum, T., and Livny, M. 2005 Distributed computing in practice: The Condor experience. *Concurr. Pract. Exper. 17*, 2–4, 323–356.

The Globus Alliance: Information Services In The Globus Toolkit 2 Release. 2013. http://www.globus.org/mds/mdstechnologybrief draft4.pdf.

Van Renesse, R. and Bozdog, A. 2004. Willow: DHT, aggregation, and publish/subscribe in one protocol. In *Proceedings of the 3rd International Conference on Peer-to-Peer Systems (IPTPS'04).* Lecture Notes in Computer Science, vol. 3279, Springer, 173–183.

Yan, W., Hu, S., Muthusamy, V., Jacobsen, H., and Zha, L. 2009. Efficient event-based resource discovery. In *Proceedings of the 3rd ACM International Conference on Distributed Event-Based Systems (DEBS'09).* ACM Press, New York.

Zhou, D. and Lo, V. 2004. Cluster computing on the fly: Resource discovery in a cycle sharing peer-to-peer system. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'04).* IEEE Computer Society, Los Alamitos, CA, 66–73.

Received January 2010; revised April 2012; accepted September 2012