Journal of Grid Computing manuscript No.
(will be inserted by the editor)

A Self-Organized Grouping (SOG) Framework for Efficient Grid Resource Discovery

Anand Padmanabhan · Sukumar Ghosh · Shaowen Wang

Received: date / Accepted: date

Abstract Dynamic and heterogeneous characteristics of large-scale Grids make the fundamental problem of resource discovery a great challenge. This paper presents a self-organized grouping (SOG) framework that achieves efficient Grid resource discovery by forming and maintaining autonomous resource groups. Each group dynamically aggregates a set of resources together with respect to similarity metrics of resource characteristics. The SOG framework takes advantage of the strengths of both centralized and decentralized approaches that were previously developed for Grid/P2P resource discovery. The design of SOG minimizes the overhead incurred by the process of group formation and maximizes the performance of resource discovery. The way SOG approach handles resource discovery queries is metaphorically similar to searching for a word in an English dictionary, by identifying its alphabetical group at the first place, and then performing a lexical search within the group.

Because multi-attribute range queries represent an important aspect of resource discovery, we devise a generalized approach using a space-filling curve in conjunction with the SOG framework. We exploit the Hilbert space-filling curve's locality preserving and dimension reducing mapping. This mapping provides a 1-dimensional grouping attribute to be used by the SOG framework. Experiments show that the SOG framework achieves superior look-up performance that is more scalable, stable and efficient than other existing approaches. Furthermore, our experimental results indicate that the SOG framework has little dependence on factors such as resource density, query type, and Grid size.

Keywords Grid computing · peer-to-peer systems · resource discovery · multi-attribute range queries · self-organization · space filling curve

A. Padmanabhan, S. Wang
CyberInfrastructure and Geospatial Information Lab and National Center for Supercomputing Application (NCSA)
University of Illinois at Urbana Champaign
Urbana, IL 61801
E-mail: apadmana@uiuc.edu, shaowen@uiuc.edu

S. Ghosh
Department of Computer Science
The University of Iowa
Iowa City, IA 52242
E-mail: ghosh@cs.uiowa.edu

1 Introduction

Grids [13] and peer-to-peer systems are increasingly being used as large-scale resource sharing environments. Scientists use these distributed computing environments to conduct knowledge discovery in their specific domains. Key services like resource reservation and task scheduling can be built only when resources can be located. Grid resource discovery service helps locate resources to meet user requirements.

Current Grids provide a distributed infrastructure that enables the assembly of numerous resources across multiple administrative domains, to create a virtual supercomputer that can be used to accomplish a specific task. This assemblage of distributed resources is dynamically orchestrated to support coordinated resource sharing through Grid middleware. The coordinated sharing of resources takes place within formal or informal consortia of individuals and/or institutions that are often referred to as Virtual Organizations (VO) [14]. Individual resource providers manage the infrastructure at their sites, making the Grid highly heterogeneous and without central authority. Furthermore, Grid resources can be either static or dynamic. Static resources have attributes that are relatively stable (e.g. OS version), while dynamic resources represent more volatile attributes (e.g. free memory, available disk space). These properties suggest that any successful resource discovery approach should be sufficiently nimble to adapt to its environment. This paper presents an approach to handle these challenges effectively by developing a self-organizing process. While we concentrate on developing a solution for discovering dynamic resources, the solution is trivially suitable for static resources too. A typical Grid node is composed of multiple resources, both static and dynamic (e.g. a node has a CPU type, an OS installation, memory and disk-space available, wallclock time) and a typical job places multiple constraints on resource availability. A Grid resource discovery system is required to handle complex multi-attribute range queries (constraints) within dynamically changing environments.

Developing an efficient and scalable solution for Grid resource discovery poses a challenge because of the:

1. dynamic nature of Grid resources;
2. absence of central authority;
3. heterogeneous and large scale nature of Grid environments;
4. unpredictability of faults; and
5. difficulty in handling complex multi-attribute range queries.

To address these challenges, a resource discovery solution must exhibit the following important characteristics:

1. support for intermittent resource availability;
2. independence from any centralized/global control and global knowledge;
3. support for attribute-based discovery;
4. support for multi-attribute range queries;
5. scalability in terms of number of users and resources, and type of resources; and,
6. provide excellent query performance.

The framework presented in this paper is designed to espouse each of these characteristics. In particular, our research aims to addresses the challenges of dynamic resource discovery and multi-attribute range query. The Self-Organized Grouping (SOG) framework [28] is designed to enable discovery of dynamic Grid resources and to handle multi-attribute range queries using space-filling curves (SFC). This framework provides a scalable mechanism to support efficient Grid resource discovery.

In the SOG framework, resources get self-organized into groups according to similarity of statistical characteristic. Previous research in small-world and unstructured P2P network shows that performance can be improved by exploiting common interest shared by users [7], and by clustering peers together [20]. In the SOG framework, we introduce the concept of grouping from a different angle – instead of grouping users with similar interests, we organize similar resources into groups. Each group is managed by a leader, which is the group representative and consists of members that serve as workers. This approach provides the framework with a unique capability to exploit the strengths of both structured and unstructured P2P networks. This along with self-organizing design of the framework achieves scalability, intermittent participation, local control, there by being resilient in complex Grid environments. Section 2 relates this work to earlier research in the field of resource discovery in distributed environments. Section 3 presents the architectural design of this SOG framework . Section 4 discusses the algorithms that implement the SOG framework, and explains how to couple SFCs and SOG framework to address the problem of multi-attribute range queries and to achieve efficient Grid resource discovery. Section 5 analyzes the SOG framework and together with a suite of experiments in section 6, we demonstrate that SOG provides a scalable and efficient solution for problems associated with Grid resource discovery. We conclude this paper with a discussion and pointers to future work.

2 Resource Discovery Approaches for Distributed Environments

Current solutions to Grid resource discovery fall broadly into two broad categories, centralized and decentralized. Centralized approaches have good lookup performance, but they scale poorly in size. Decentralized approaches are generally scalable in terms of the number of nodes that can be handled, but suffer from a significant deterioration in average lookup performance.

In peer-to-peer (P2P) systems, decentralized methods can be divided into two major categories: unstructured and structured. Unstructured P2P network (e.g., Gnutella [32]) generally communicates using broadcast (e.g. flooding), and consequently incur significant communication overhead while finding existing resources cannot be guaranteed. Distributed hash tables (DHT) based structured P2P networks (e.g., CHORD [36], CAN [31]), are often suitable for looking up existing data items and guarantee query responses within a small number of hops. However, a dynamic resource environment results in a substantial number of updates, which likely causes dramatical increase in the overhead of maintaining the DHT. Furthermore, DHT requires every resource-value pair to have a unique key, which may limit its scalability if used for Grid resource discovery. SOG captures the strengths of both structured and unstructured approaches while avoiding their limitations by espousing the self-organization capability of fully unstructured architecture and adding a lightweight structure to obtain efficient query performance. The SOG approach has some similarities with the super-peer network [38], in that both approaches form a cluster of peer nodes. But unlike super-peer networks SOG leaders do not have to store an up-to-minute status about client resources.

In Grid resource discovery research, various approaches have been studied as well. Condor Matchmaker [30] follows a centralized approach and maintains a central server responsible for matching the advertisements between resource consumers and resource providers. A fully-decentralized approach based on different request forwarding schemes have been studied by Iamnitchi et al [15], while a DHT based structured decentralized approach was adopted by Zhu et al [39]. A hierarchical structure, formed by GIIS (Grid Index Information Service) and GRIS (Grid Resource Information Service), is employed by the Monitoring and Discovery Services (MDS) [8] from the Globus Toolkit. We believe that the architecture and technique developed in SOG is orthogonal to MDS efforts, and could be integrated into the MDS infrastructure. Mastroianni et al. [21] adapted the super-peer network to perform resource discovery. A bio-inspired approach relying on swarm intelligence using self-organizing agents with an ant-based replication and mapping protocol was studied by Forestiero et al. [10] for So-Grid [11]. Foster et al [12], forecast the convergence of of strengths of Grid and P2P technologies, which has spurred the development of many P2P inspired algorithms for handing challenging Grid research issues including information systems [34], service [6] and resource [22] discovery.

Multi-attribute range queries for dynamic resources have been addressed in Mercury [4], SWORD [27], XenoSearch [35]. All these methods are based on DHTs. Mercury creates a hub for each attribute considered and replicates data items on each hub. A query is forwarded to one of the hubs based on the selectivity of the query. XenoSearch creates a separate DHT instance for each attribute, while creating its query routing structure explicitly rather than using built-in DHT successor pointers. In addition, XenoSearch provides approximate answers using Bloom Filters [18] and supports DHT-based multi-attribute range queries by directing them to each separate dimension and aggregating results. SWORD's *multi-query* approach is similar to XenoSearch, while their *single-query* approach is similar to Mercury. SWORD also allows users to define inter-node or edge attributes that are absent in other approaches. Prefix hash table (PHT) [29] and other a trie-like data structure [6], offers an alternative range search strategy based on tries built on DHTs. In contrast to these methods, the SOG method does not replicate either queries (as in XenoSearch) or data elements/updates (as in Mercury).

Space-filling curves have been used in a variety of domains to handle the mapping between multi-dimensional space and one-dimensional curves. [37] uses SFCs to effectively locate services on a mobile self-organizing network. In particular [33] and [1] used SFCs to address multi-attribute queries. However, unlike the SOG framework, [1] uses an inverse SFC mapping (*1*-dimension to d-dimension) to map a resource to peers in the CAN overlay based on a single resource attribute. [33] uses a Hilbert SFC mapping from d-dimension space to 1-D index space to form a Chord overlay network. Also, the work in both [33] and [1], primarily deal with keyword search. Unlike these, SOG is not based on DHTs, and thus avoids the problems of costly updates involved in the publication of dynamic resource information.

3 Self-Organized Grouping (SOG) Framework

In this section we present the basic building blocks of the SOG architecture. We explain how the self-organized grouping approach handles dynamically changing Grid resources; and how space-filling curves are used in SOG to address multi-attribute resource discovery requests.

The SOG framework 1) supports intermittent resource participation, 2) works exclusively with local knowledge, and 3) functions based on complete local control. The application of a space-filling curve to handle multi-attribute range queries coupled within SOG 1) exploits locality preserving mapping to facilitate multi-attribute range queries, 2) assembles resources into a limited number of dynamic groups using different order SFC mappings, and 3) handles resource heterogeneity by grouping resources with equivalent SFC mappings.

3.1 Overlay Network

In the SOG architecture, a two-layer overlay network is formed to maintain groups and process queries. The lower layer is constructed using a lightweight gossip protocol [3]. This layer, each node of which has $O(\log n)$ neighbors [17], is primarily used to forward gossip messages (for leader election).

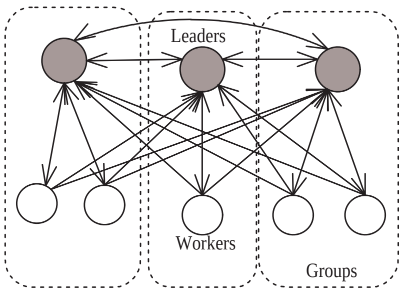

Fig. 1 The upper level graph maintained by SOG (from [28])

The upper layer (figure 1) is primarily used for query handling. It represents the interconnection among leaders and workers and is designed as follows: 1) each node has a link to all leader nodes; 2) each leader node has a link to all of its own worker nodes. The resulting graphs can range from a star graph on one extreme to a fully connected graph at the other. Neither of these extreme cases is desirable in practice. The star graph results when there is a single group (this is equivalent to a single/centralized server architecture), while the fully connected graph occurs when every node is a group in itself. Graph connectivity increases as the number of groups (and consequently the number of leaders) increases. Graph diameter is kept as two, thereby helping achieve excellent performance for query processing.

A typical upper layer graph maintained by SOG lies between the two extremes described above. The SOG overlay is initialized by specifying a similarity threshold that controls the number of groups. The trade-off that guides the threshold specification is determined by the balance number of groups and the size of each group. If the number of groups is too small, then the leader nodes of these groups might be forced to handle an excessive amount of loads, since the group sizes tend to be large, and this will lead to performance bottlenecks. On the other hand if there are too many groups, the storage cost increases since each node store pointers to all the leader nodes and so does the cost of communication required to form and maintain the groups. By striving to balance the trade-off between group sizes and the number of groups, we ensure the grouping approach deployed by SOG remains scalable.

3.2 The SOG Architecture

The SOG architecture is based on the principle that nodes in the Grid are organized automatically into groups according to their resource's statistical characteristic that facilitates the organization and usage of this framework.

Self-organized groups are formed in a bottom-up fashion. Each node starts as a group by itself. Through the execution of SOG components, groups merge into larger groups guided by the similarity threshold of statistical characteristic. This SOG architecture focus on the development and evaluation of self-organized grouping capability. This capability is implemented by the SOG components (algorithm 1) that are executed on each node in a non-deterministic fashion.

Algorithm 1 SOG Components

```
1: Publish resource information to leader periodically if leader
   exists;
2: Handle group forming and maintenance;
3: Receive and process system messages (like subscribe, proclaim
   leader, conflict);
4: Receive and process query messages;
```

An important strategy of SOG is to manage resource dynamics through grouping, and thereby to limit the amount of dynamic information being published. The only active dynamic information being published to enable SOG framework originates from a worker and is directed to its leader. This helps limit the communication overhead incurred by the SOG framework. This scheme enhances SOG scalability by limiting the volume of dynamic information being published without any negative impact on query performance.

Another task handled by the SOG framework is the creation and maintenance of groups. Group checking takes place periodically to ensure that each node belongs to an appropriate group as resource status dynamically changes. When a new node joins the Grid, it begins collecting temporal samples of resource values. Once it has sufficient temporal samples, it searches for and if successful, joins an existing group, the statistical characteristic of which is similar to itself. A group's statistical characteristic is the same as its leader's characteristic that was gossiped when the leader was elected. If such a search fails, the newly joining node creates a new group with itself as the group leader. This new group evolves with other existing groups. The grouping of statistically-similar nodes increases query performance of resource discovery by decreasing the number of nodes that need to be visited by a query.

Using the overlay network, SOG implements a communication framework that provides all nodes with the capability to receive, process, and send messages and forms a backbone on which query processing is handled. This availability of information on group leaders and their characteristic allows the SOG framework to direct a query to an appropriate leader i.e. a leader in charge of a group with resources similar to the ones requested by the query. The leader, in turn, can direct the query to one of its workers that is able to satisfy the query.

The SOG querying mechanism limits a query's search space from the entire Grid to a group through a single step and resource location inside the group in the next step. This process is metaphorically similar to searching for a word in an English dictionary by identifying its alphabetical groups at the first place. If the querying mechanism is unable to identify an ideal group or if such a group does not exist, it becomes necessary to forward the query to another group leader that may satisfy the query. This forwarding mechanism between group leaders achieves high resource discovery efficiency by keeping resource discovery scope at the group leader level. In case no leader can be identified, queries are randomly forwarded to a gossip neighbor.

Gossip messaging is expensive, therefore, to keep the number of gossip messages to a minimum while efficiently handling resource discovery queries, SOG treats group leaders differently than other resources. More specifically, new replacement group leaders are introduced to maintain major information for the existing groups, as the current group leaders leave. This strategy avoids dissolution of a group and thus saves gossip messages that would be required without the application of the strategy. In addition, leader election is based on the evaluation of group local information. Multiple nodes may contend to become leaders even though they should belong to a single group. This case is defined as a conflict condition and only one leader is allowed in the current SOG design.

In essence, self-organized groups are designed to have the following three properties.

1. Each group is dynamically formed and maintained and includes a collection of nodes with a single representative node elected as its leader.
2. The nodes within each group are statistically similar based on a specified resource characteristic. The selection of specific resource characteristic is guided by the understanding of common resource requirements from applications that issue resource discovery queries.
3. The size of each group is indirectly determined by a similarity threshold. This threshold is chosen based on the number of Grid resources, resource density, query load, and specifies a maximum range within which resource characteristic are deemed similar. Generally, the larger the threshold, the larger group sizes tend to become.

These properties lead a query to find the appropriate group leader as the first step, and then searching within the group as the next step.

3.3 Using Space-Filling Curves to Handle Multi-Attribute Range Queries

While describing the SOG architecture we mentioned that groups are formed by a collection of nodes with similar resource characteristics. For simplicity, this could be thought of as based on a single attribute from resources. However, there is no constraint placed by the SOG framework that limits the statistical characteristic to be derived from a single attribute. In fact the statistical characteristic, may be a function based on some or all of the attributes involved. Though it is possible to handle a multi-attribute range query by just using a resource characteristic from a single attribute, selecting a suitable attribute can be extremely challenging. This is because the selection needs to be based on the characteristics of Grid environments and application requirements, both of which are expected to be dynamic. Empirical evidence confirm a performance deterioration when resource discovery query is not against the grouping attribute [28]. This suggests a need to use multiple-attributes for evaluating the resource characteristic. One such possible function can be based on space-filling curves [2], which maps an n-dimensional space into a 1-dimensional curve. Here we consider the Hilbert space-filling curve to illustrate how space filling curves are used to evaluate the statistical characteristic.

3.3.1 Hilbert Space-Filling Curve

A space-filling curve provides a mapping between 1-D and d-dimensional space. This mapping can be thought of geometrically, dividing d-dimensional space into d-dimension hypercubes, with a line passing exactly once through each hypercube. This line transforms coordinates from a d-dimension space to the corresponding 1-D space that define their relative ordering (figure 2 illustrates first, second and third order Hilbert curves in the 2-D space). The Hilbert space-filling curve [23] is one of space-filling curves the is most widely used.

The locality preserving property of Hilbert SFC results in coordinates that have close mapping in 1-D space are guaranteed to be close in d-dimensional space. However, this property does not hold in reverse mapping, because two coordinates that are close in d-dimensional space might not be close in its corresponding 1-D space. So if the 1-D Hilbert SFC mapping is used to form SOG groups, two resources that could have been otherwise grouped together are in separate groups. However, SOG is designed to overcome this drawback and nd a small increase in the number of groups should not have a noticeable effect on query performance.

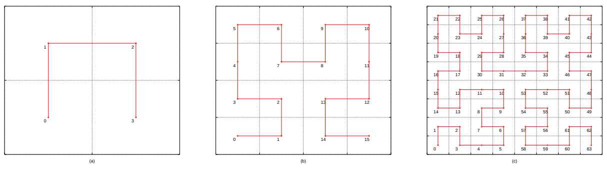

Fig. 2 2-D Hilbert space-filling curve (a) first-order; (b) second-order; (c) third-order.

In this work, an iterative implementation is used to construct Hilbert SFC based on Butz's algorithm [5], along with the improvements presented in [19]. The use of a faster implementation [16] is planned for future work, which should not affect the research findings being presented in this paper. Using Hilbert SFC also allows our approach to flexibly control the number and size of groups. The number of iterations in this algorithm is equal to the order of the Hilbert curve being constructed. The order of the curve represents the granularity of the curve. A d-dimension order 'k' Hilbert SFC, will partition the d-dimensional cube into 2^{kd} sub-cubes. We choose the 1-D index mapping from the Hilbert SFC to measure resource similarity. The SOG framework groups resources autonomously by taking advantage of the clustering property [23] of Hilbert SFC. We exploit the locality preserving mapping to perform grouping around a 1-D index space.

3.3.2 Motivation for using Hilbert Space-Filling Curve

In this section we try to motivate the reason for our choice of Hilbert SFC and address concerns a reader may have about the wisdom of using Hilbert SFC for determining SOG Groups. Instead of using SFCs, one might believe measuring similarity directly on the d-D Euclidean space, or for that matter using any function that maps the resource space to a group index, would be a easier option. In fact, as we have stated earlier, the SOG approach is agnostic with respect to the metric used to measure similarity. This, we argue, is one of the advantages of the SOG approach and makes it adaptable to different scenarios. Even though any similarity metric can be used, we do believe that using SFC for similarity measure is appropriate for dealing with multi-attribute range queries due to the following reasons:

1. Similarity measurement in a 1-D Hilbert space is simple and fast;
2. Amount of storage is needed across all nodes is low; and
3. Communication cost is low.

Faster similarity measurement is important because it is one of the widely used operations in the SOG framework. It is used in algorithms to determine resource grouping, detecting leader collision and query routing. On the Hilbert space, identifying similarity between two Hilbert-codes is a matter of doing a simple XOR operation, while the same operation on a d-D Euclidean space will be much more expensive[1]. An astute reader will point out that calculation of the Hilbert code itself will incur cost. This is indeed true, but it is important to note that once a Hilbert code has been calculated, it is used a number of times to measure similarity since this operation is occurs much more often[2].

Since every node in the system records the information about all the leader nodes, it is also important to minimize the amount of data that is needed to be stored to identify the leader and describe its characteristics. The use of Hilbert curve helps us limit the amount of space needed to describe the resource. For example, a curve with dimension d and order k required only k*d number of bits to describe the resource characteristics[3]. Since the resource can be described in k*d bits, this lowers the overall storage requirement needed by each node. Furthermore, by sending the 1-D Hilbert code to describe resources, the quantity of data to be communicated between the nodes is reduced, lowering the requirement of network bandwidth.

One may argue that similar savings may be achieved by normalizing the resources and assigning a group index. Assigning a group index alone will not be sufficient since the characteristics of the group is necessary for routing, so some sort of mapping between the index space and resource space is needed. In our implementation the Hilbert SFC provides such an implicit index, based on an locality preserving mapping from index to resource space. Any approach that uses an implicit mapping will predefine the logical group boundaries and will be no better than the Hilbert SFC, since

[1] Requires d-subtraction, d-multiplication, d-addition and one square-root calculation.

[2] Similarity measurement occurs multiple times during every leader election, query routing and comparison steps.

[3] In our experiments this comes out to 16 to 40 bits.

it is not possible to guarantee closeness in d-D space will result in closeness in 1-D space. If a method with an explicit dynamic mapping or no mapping at all (i.e. d-D Euclidean coordinate) is used, the amount of data that needs to be stored and communicated will substantially increase. In a similar vein, a normalization scheme will not perform better than a SFC. Furthermore, the SFC provides a structured and well defined code, which unlike an unstructured normalization and mapping scheme, can be efficiently computed and is well understood.

Lastly, we would argue that even if in some cases replacing SFC with a d-D coordinate space is a possibility, the gains are quite minimal. The only potential advantage is that the resources may be found faster (i.e. smaller number of hops). But as we will see in our experiments (section 6), the performance of SOG as measured by the number of hops is excellent and we do not anticipate any significant performance improvements even if we would decide to do a direct comparison in the d-D space. In the next section, we discuss in detail how this Hilbert SFC mapping is used for creating SOG groups and how multi-attribute range queries are handled.

4 Algorithms for SOG implementation

In this section we present the algorithmic details of SOG framework. We discuss in detail how self-organization occurs through group formation, how queries are handled and how Hilbert SFC mappings is used to handle multi-attribute range queries.

4.1 SOG Algorithm

4.1.1 Data and Query Model

Each resource is represented as a typed (attribute, value) pair. Each node is composed as a conjunction of a number of such resources. Each resource is represented as a tuple of (***type, attribute, value***), where a type can be ***int, float, double, char, or string*** in the current implementation.

Similarly, a query in SOG is composed of a conjunction of tuples in the form of ***(attribute, operator, value)***. The type associated with the attribute in a query is assumed to be same as that of the corresponding resource. The operators currently supported include $<$, $>$, $\leq$, $\geq$, $=$, $\neq$. Disjunction queries can be split and implemented as separate queries.

4.1.2 Data Structures

For each node n we define:

```
1: $c$; {Statistical characteristic}. Initially $c \leftarrow undefined$
2: $R$; {Set of recent observations}. Initially $R \leftarrow \emptyset$
3: $L$; {Set of all leaders}. Initially $L \leftarrow \emptyset$
4: $S$; {Set of statistical characteristic ($c$) values for all $L$, addressable as $S(l')$, where $l' \in L$ }. Initially $S \leftarrow \emptyset$
5: $l$; {Node's Leader}. Initially $l \leftarrow undefined$
6: $isLeader$; {Is node a leader?}. Initially $isLeader \leftarrow false$
7: $x$; {Rounds after which the statistic is recalculated}. Initially $x \leftarrow pre-defined\ constant$
```

Additionally for leader nodes we define:

```
1: $W$; {Set of all worker nodes that report to it}. Initially $W \leftarrow \emptyset$
2: $S'$; {Set of $c$ values for all $W$, addressable as $S'(w)$, where $w \in W$}. Initially $S' \leftarrow \emptyset$
3: $V$ {Set of current resource status for all workers in $W$, addressable as $V(w)$, where $w \in W$}. Initially $V \leftarrow \emptyset$
```

4.1.3 Subscribe and Unsubscribe

A node n joins the Grid through node n', which it learns about offline. The node n sends a 'subscribe' message to n', which forwards the message to a subset of its neighbors (algorithm 2). It receives the current state of the Grid as captured by node n'. Furthermore, node n will receive all the gossip messages that n' will receive in future.

Algorithm 2 Subscribe(n,n'): Node n joins the Grid through node n'

```
1: Send a subscribe message to $n'$ {Subscribe as in [3]}
2: Receive sets $L_{n'}$ and $S_{n'}$ from node $n'$
3: $L \leftarrow L \bigcup L_{n'}$; $S \leftarrow S \bigcup S_{n'}$;
```

Algorithm 3 deals with the case in which a node leaves the Grid. If the departing node happens to be a leader, then a replacement leader is selected before the node unsubscribes. This prevents the loss of existing grouping knowledge.

4.1.4 Group Formation and Maintenance

Algorithm 4 outlines the formation and maintenance of self-organized groups. Each node periodically checks to see if it is placed in an appropriate group, based on its

Algorithm 3 Unsubscribe(n): Node n leaves the Grid

```
1: if $n.isLeader = true$ then
2:   replace_leader() {Select replacement}
3:   Piggyback $resign$ message to unsubscribe message
4: end if
5: Unsubscribe as in [3]
```

temporal values and the statistical similarity measure used (steps 5-7, algorithm 4). If not, it searches for a group to which it should belong (step 9). If there is no existing group for the node to join, it creates a new group with itself as leader and its current characteristic as the group's characteristic (step 12). If the departing node happens to be a group leader, then it will first select a replacement leader before it leaves the group, thereby maintaining the existing grouping information (step 21).

Algorithm 4 Group Formation, Maintenance: at node n

```
   $event\_count \leftarrow 0${Timer count for periodic recalculation of statistic (c)}
1: r $\leftarrow$ poll_resource();{Periodic resource status sampling}
2: Record_entry(R, r);{Add current sample to R, replacing old samples in round-robin fashion if necessary}
3: event_count++;
4: if $event\_count = x$ then
5:   c $\leftarrow$ recalculate_c(R);{Recalculate characteristic (c)}
6:   event_count $\leftarrow$ 0;
7:   if $l = undefined \vee (n.isLeader = false \wedge is\_not\_similar(S(l), c))$ then
8:     {Either node $n$ belongs to no group or $n$ is a non-leader that has moved away from its group}
9:     $l' \leftarrow find\_leader\_node(c, L, S)$;{Find an appropriate leader}
10:    if $l' = undefined$ then
11:      {No leader was found}
12:      random_delay() {This delay is added to reduce conflicts (see analysis section)}
13:      gossip_proclaim_leadership(n,c);{Proclaim node as leader through gossip}
14:    else
15:      {A new leader was found}
16:      Inform node $l'$ about $n$ joining its group
17:      Inform node $l$ about $n$ leaving its group if $l \neq undefined$
18:      $l \leftarrow l'$ {Assign new leader}
19:    end if
20:  else if $n.isLeader \wedge is\_not\_similar(c, S(n)) \wedge change\_persistent()$ then
21:    {Leader node $n$ has moved away from its group range}
22:    replace_leader();
23:  end if
24: end if
```

The strategies developed to minimize the number of gossip messages are two-fold: 1) a leader changes its group only if it observes a persistent change in its statistical characteristic; 2) once a node has become a leader, it does not gossip its changing characteristic periodically. Through the application of these strategies, a group is identified by the characteristic, that was gossiped when its last leader was elected. When a leader's current characteristic is significantly different from its group's characteristic (steps 19-22), the leader identifies a node among its workers whose characteristic is the closest to the group's characteristic and instructs that node to take over as the new leader of the group (steps 1-5, algorithm 5). This method of selecting a replacement leader ensures that the group under the new leader will have a similar characteristic to the group's characteristic under the current leader and there will be a minimal change in the group composition. Algorithm 5 is used to implement leader replacement. Once a replacement leader has been found, the current leader identifies an appropriate group for itself (steps 6-18).

Algorithm 5 Replace Leader: replace_leader() at node n

```
1: Select a node $l'' \in W - \{n\}$ | similarity_value($S'(l'')$ , $S(n)$) is minimum {Find a node with characteristic most similar to the group's characteristic}
2: if $l'' \neq undefined$ then
3:   Send a message to $l''$ asking it to proclaim leadership
4:   $\forall w \in W$, send a message identifying $l''$ as a new leader
5: end if
6: if $n$ is not unsubscribing then
7:   {If a node is unsubscribing it does not need to find a group it should belong to}
8:   $l' \leftarrow find\_leader\_node(c, L, S)$;{Find an appropriate leader}
9:   if $l' = undefined$ then
10:    {No leader was found}
11:    random_delay() {This delay is added to reduce conflicts (see analysis section)}
12:    gossip_proclaim_change_characteristic(n,c); {Proclaim new characteristic through gossip}
13:  else
14:    {New leader was found}
15:    gossip_proclaim_resign_leadership(n);{Resign leadership through gossip}
16:    Inform node $l'$ about $n$ joining its group
17:    $l \leftarrow l'$; {Assign new leader}
18:  end if
19: end if
```

Due to autonomous decisions made at the node level, a conflict occurs when multiple nodes attempt to form different groups that have statistically similar characteristic. This results in a temporary fracture of a logical group with multiple nodes claiming to leaders of a small faction. Conflict detection occurs on at-least one of the conflicting leaders. This detection is done when a leader node receives a proclaim message from another node that claims the leadership of a statistically similar group. Once a conflict is detected the leader sends a 'conflict detected' message to the offending node along the effective size of its group. The effective size of the group is same as the group size if there are no other outstanding conflicts; but if there are other outstanding conflicts, it is the group size that was reported to the first offending node. This guarantees that if there are conflicts among more than two groups exactly one leader is elected for the merged group after all conflicts are resolved. Conflicts among more than two leaders are handled in a pairwise manner. The groups then merge according to their effective size reported with the leader of the larger group becoming the leader of the merged group; and ties are broken randomly[4].

4.1.5 Query Routing

A query request is submitted to a local node. SOG is designed to return a node at which the resource is available. A non-leader node tries to satisfy a request if it can, otherwise it will direct the query to the leader of a group with the pre-specified characteristic similar to the request made in the query. In the absence of such a leader, the query is randomly forwarded to one of the node's gossip neighbors (the lower level overlay). A leader node (included in its group) will try to find a worker node, such that based on the leader's current knowledge, the worker node is able to satisfy the query. If such a worker is not available, the leader tries to find another leader that is likely to satisfy the query. The query is randomly forwarded as a last resort. Algorithms 6 and 7 handle queries at non-leader and leader nodes respectively. In section 5 we analyze these algorithms, particularly with respect to complexity of leader election process, and communication overhead. SOG resource discovery mechanisms are evaluated by experiments in section 6.

4.2 Using Hilbert SFC Mapping

In this section we illustrate how the Hilbert SFC mapping is used to create SOG groups and how multi-attribute range queries are handled. We have already discussed the motivation of using the Hilbert SFC mapping for tacking multi-attribute range query in section 3.3.2.

[4] We can use a predefined scheme (e.g. lexical ordering of IP address).

Algorithm 6 Handle Query: at non-leader node n

```
1: if n.can_satisfy_query()=true then
2:    {Query is satisfied}
3:    Notify requesting node about resource availability
4: else
5:    {Forward query, direct query to suitable leader}
6:    Find $l' \in L|is\_similar(S(l'), query\_request)$ OR
      $is\_likely\_to\_satisfy(S(l'), query\_request)$
7:    if $l' \neq undefined$ then
8:       Forward query to $l'${A Leader is found}
9:    else
10:      {No leader is found that might satisfy query}
11:      Randomly forward the query to one of the node in the
         local view formed by gossip
12:   end if
13: end if
```

Algorithm 7 Handle Query: at leader node n

```
1: Find $n' \in W|is\_likely\_to\_satisfy(S'(n')\ , query\_request)$
   {Find a worker to forward the query to}
2: if $n' \neq undefined$ then
3:    {Worker capable of satisfying query found}
4:    Forward query to $n'$
5: else
6:    {No worker node is found}
7:    Find $l' \in L|l'! = n \wedge is\_likely\_to\_satisfy(S(l')$
      $, query\_request)$ {Find another leader capable of satisfy-
      ing the query}
8:    if $l' \neq undefined$ then
9:       Forward query to $l'$ {Leader is found}
10:   else
11:      {No leader is found that might satisfy query}
12:      Randomly forward query to one of the node in the local
         view formed by gossip
13:   end if
14: end if
```

4.2.1 Forming Groups

Hilbert SFC is used to produce a unique transformation of the d-dimensional resource status information to a corresponding 1-D Hilbert coordinate section. Using this 1-D Hilbert coordinate as the grouping attribute, SOG groups can be formed in a bottom-up fashion. Each node autonomously and periodically checks to ensure that it belongs to a correct group, based on its Hilbert coordinate. If its Hilbert coordinate has changed, a node attempts to identify and get assigned to the correct group. Details on how groups are formed and managed in SOG have been discussed in the earlier section[5]. Figure 3 illustrates an example of how grouping is achieved in SOG using Hilbert coordinates. Nodes that lie in the same cell are mapped to the same Hilbert coordinate section, and are grouped together in the SOG framework. In figure 3, the nodes are arranged into three groups by the SOG grouping process.

[5] The grouping process remains unchanged with the grouping attribute replaced by the Hilbert coordinate.

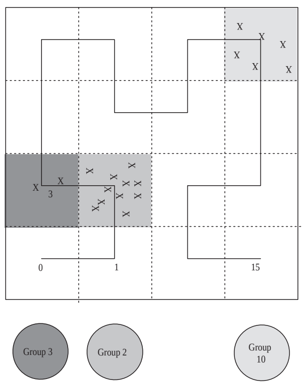

Fig. 3 Forming Self-Organized Groups using Hilbert SFC Mapping

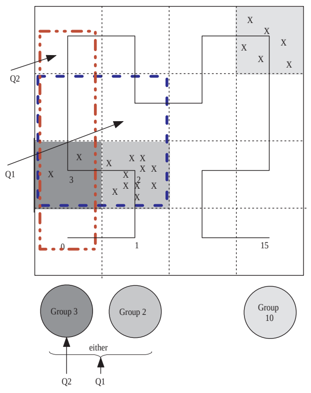

Fig. 4 Multi-Attribute Range Query Routing

Using the 1-D Hilbert coordinate as the grouping attribute also allows SOG to control group sizes through the 'order' of the Hilbert curve. By changing the order of the Hilbert SFC used, we can control the number of groups and the size of each group. A high order curve results in a larger number of groups (with each group having relatively a few members), while a low order curve would result in a smaller number of groups. In practice a very high order curves are not necessary for the sake of grouping in SOG [6]. Furthermore, this space tends to be sparsely populated with resources due to resource clustering and only a small fraction of the potential groups tend to form. In our experiments curves with at most order of 4 were evaluated.

4.2.2 Multi-attribute Range Query Processing

SOG query processing component routes queries on its overlay, by exploiting the information on the identity of group leaders and their Hilbert coordinates[7]. Each node that receives a query is able to route it to a suitable leader, which represents a group of workers, one of which is likely to satisfy the query. When a multi-attribute query arrives at a leader node, the leader routes the query to a worker that is capable of satisfying the query. When such a query is received by a non-leader node, it first checks if it can successfully satisfy the query. If not, the processing node identifies a set of groups such that the intersection between the groups space and the query space is not empty. The query is forwarded to any one of the groups in this set, since all the groups in this set are likely to be of interest to the query. If the intersection set is empty, the query is randomly forwarded on the lower-level overlay layer. This intersection between the resource and query space is determined by comparing the Hilbert coordinates.

Figure 4 illustrates the process of routing the query. The queries Q1 and Q2 are interested in ranges as shown in figure 4. Q1's query space is mapped to sections 2, 3, 4 & 7 on the 1-D Hilbert coordinate space. Since groups associated with both sections 2 & 3 exist, the query is routed to the leaders of either of the groups with equal probability. Q2's' query space on the other hand transforms to Hilbert coordinate sections 0, 3, 4 & 5. The query is routed to the leader of group 3 since this is the only existing group, for which the intersection between group space and query space is not empty. A time-to-live (TTL) parameter is also set, so that a query can be dropped after a certain number of hops on the overlay, if resource does not exist, or cannot be successfully discovered.

We anticipate only a relatively small number of attributes for grouping, perhaps in the 10's. A fair question would be what if we have 1000's of attributes in the system that could be potentially queried. In general there are a small number of attributes (e.g. memory, disk availability, wallclock limit) that tend to be most important and widely used and such attributes are well suited to be used for grouping in the SOG framework. The choice of particular attributes for grouping would depend extensively on the nature of Grid environments and their usage. It should be noted that using too many attributes for grouping may cause the space to be finely divided and the grouping of resources will not be achieved. If on the other hand we use too few attributes for grouping then searching on the remaining attributes would suffer performance deterioration. Since both options could have potentially negative impact on query performance, we try to achieve a balance between these conflicting conditions by selecting widely used attributes for grouping. Even attributes that have not been used for grouping can still

[6] Even with an order 4 curve on a 4-D space can have potentially 2^{4*4} groups, while the 10-D space could produce 2^{4*10} groups.

[7] The query processing remains unchanged except for the use of the Hilbert coordinate as grouping attribute.

be searched, though the performance of queries against these attributes might suffer. Experiments in the section 6 help us verify the performance of the SOG framework.

5 Analysis of the SOG Framework

In this section we evaluate the SOG framework and demonstrate its low communication overhead and quick self-organization, associated with the leader election process. To do this we calculate the efficiency of the leader election algorithm in terms of the number of messages needed and time taken for leader election. The problem of leader election that we address is different from the traditional leader election problem, in that not only all the members of a group should agree on one leader, but also all the nodes in the system need to know the leader's identity. We concentrate on 'gossip' messages to assess communication overhead since they account for the majority of the overhead. Since leaders play a pivotal role in handling queries efficiently, the election of a leader is used as a metric to measure self-organization. Also time taken for leader election indicates how well the framework handles system perturbations (e.g. resource characteristic changing, leader leaving).

5.1 Model

There are n distinct nodes, which form g groups, with m nodes in each group. All decisions are made autonomously and independently at each node. We analyze the number of messages needed to elect a leader for each group. The model assumes no loss of messages. Furthermore, none of the nodes in the system have a global knowledge. Instead, each node maintains a local view of $O(\log n)$ neighbors, based on the way the lower-level overlay is formed (section 3.1). Moreover, group membership is not known beforehand and is dynamically determined by a statistical characteristic, so traditional methods of leader election (e.g. selecting node with the highest id) are not feasible for our problem. We consider the following cases in our analysis:

1. Worst case analysis (No suppression delay)
2. With a random suppression delay added

5.2 Worst Case Analysis

Lemma 1 *The worst case message complexity for electing the leader of a single group is $O(m)$ gossip messages, where m is the size of the group.*

Proof We will first consider the group containing m nodes trying to elect a leader. In the worst case all the nodes will 'proclaim' themselves as leader. Hence $O(m)$ gossip messages[8] will be sent, one by each node. This will be followed by conflict resolution, done pairwise, leading to $2\times\binom{m}{2}$ messages and $m-1$ gossip messages to resign leadership, before a leader is elected.

Hence leader election involves, $O(m)$ gossip and $O(m^2)$ pairwise messages. Since each gossip message results in $n \log n$ and $m \leq n$ $O(m)$ gossip messages is a dominant factor. Hence the proof. □

Theorem 1 *The worst case message complexity for electing all leaders is $O(n)$ gossip messages, where n is the total number of nodes.*

Proof Since each node in the system will occur exactly in one group, proof follows from Lemma 1 □

We could do better by simply sending direct messages from each of the m nodes, to the n nodes in the system. This would lead to just $O(m * n)$ messages, which would be better than the worst case presented above. But such an approach is infeasible, because as stated earlier no node in the system has the knowledge of all n nodes in the system. We will discuss a technique to improve this message complexity in the next section.

5.3 Analysis - with random suppression delay added

Suppression is a technique by which we introduce some random delay before a node decides to send a 'proclaim' message. That implies that before a node sends a 'gossip' message it waits for some random time and then revisits its decision to send the 'gossip' message. This is a standard technique and has been used in previous research [9] for leader election analysis. Let T_d be a random variable which has a probability density function $p(t)$ and the corresponding cumulative distribution function $P(t)$. Let Δ be the propagation delay, i.e. the amount of time taken before a message sent by gossip reaches all the nodes. We will consider a bounded propagation delay, i.e. $\Delta = constant$[9].

Let $\Delta = \theta$, a constant, i.e. if a message was sent by a node at time t all nodes in the system will have received the message by time $t + \theta$. The goal is to minimize the time taken for leader election to complete as well as minimize message complexity. We will construct the function $p(t)$ as a discrete uniform distribution defined

[8] One gossip message will result to $n \log n$ individual messages.
[9] In practice this constant could be equal to 3 * maximum round trip time between grid nodes.

as follows

$$p(t) = \begin{cases} \frac{1}{m} & \text{for } t \in 0, \theta, 2\theta, \ldots, (m-1)\theta \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

At this stage one may argue that this function cannot be constructed by the nodes, since the nodes have no knowledge of m. It is true that nodes do not have a correct knowledge of m, but since we know that each node has $O(\log n)$, each node can essentially estimate n, which can be used in place of m in the above function[10].

Since $p(t)$ is the probability function used for suppression delay, nodes can send gossip messages at time slots $t = (0, 1, \ldots, m-1)\theta$. If one or more nodes send 'proclaim' messages at $t = c\theta$, none of the other nodes will send messages at $t \geq (c+1)\theta$. This is because the bounded message propagation delay of θ guarantees that all nodes will have received the message at $(c+1)\theta$, and based on the algorithm no other nodes will send a 'proclaim' message beyond this time. Since we have established the number of gossip messages is a dominant factor, the pair-wise messages needed to resolve conflicts, can be neglected.

5.3.1 Calculating Time Complexity of Leader Election

:

This section estimates the time taken for the leader of a group to be elected. We will show that the expected time for electing a leader is $O(1)$ and that with high probability the time taken to elect a leader is $O(\ln m)$.

Lemma 2 *A leader of a group is expected to be elected in $O(1)$ time.*

Proof Let X be a random variable which denotes the earliest slot at which a proclaim message was sent. We will like to show that $E(X) = O(1)$;

$$P[Node\ sends\ `proclaim'\ at\ time\ t = k\theta] = \frac{1}{m}$$

$$P[No\ `proclaim'\ at\ time\ t = k\theta] = (1 - \frac{1}{m})^m \approx \frac{1}{e}$$

$$Prob(X = i) = \begin{cases} \frac{1}{e^i}(1 - \frac{1}{e}) & \text{for } i = 0, 1, 2, \ldots, \\ 0 & \text{otherwise} \end{cases}$$

So X essentially has a Geometric distribution with parameter (success probability) $1 - \frac{1}{e}$

$$\therefore E(X) = \frac{\frac{1}{e}}{1 - \frac{1}{e}} = O(1)$$

So a group leader is expected to be elected in constant time. □

[10] Since $n \geq m$ it provides a upper bound value for m.

Lemma 3 *With high probability the leader of a group will be elected in $O(\ln m)$ time.*

Proof Let X be a random variable as defined in lemma 2. We will like to show that $Prob(X \leq c \log m) > 1 - \frac{1}{m}$. Also as we showed in lemma 2 X has a Geometric distribution with parameter $1 - \frac{1}{e}$ and a probability density function of

$$Prob(X = i) = \frac{1}{e^i}(1 - \frac{1}{e})$$

So,

$$Prob(X < k) = \textstyle\sum_{i=0}^{k-1} \frac{1}{e^i}(1 - \frac{1}{e})$$

$$> \quad 1 - \frac{1}{e^{k-1}}$$

Using $k = \ln m + 1$,

$$Prob(X < \ln m + 1) > 1 - \frac{1}{m}$$

Hence the proof. □

With these two lemmas we have shown that the leader of a single group is elected with high probability in $O(\ln m)$ time, and the expected time for leader election is constant. Since stabilization in SOG occurs on election of group leaders, the system does not take too long to stabilize.

5.3.2 Calculating Message Complexity of Leader Election

In this section we estimate the number of messages sent before a group leader is elected. We will show that the expected number of gossip messages[11] taken to elect a leader is $O(1)$ and that with high probability the number of gossip messages taken is bounded by $O(\frac{\ln m}{\ln \ln m})$.

Lemma 4 *A leader of a single group is expected to be elected with $O(1)$ gossip messages.*

Proof Let Y be a random variable which represents the number of nodes that decide to become leader at any slot. So in order to prove this lemma all we need to show is that $E(Y) = O(1)$. This is true since each node selects a slot using the discrete uniform distribution (probability density function in equation 1). Hence proof. □

Lemma 5 *With high probability the leader of a single group will be elected with $O(\frac{\ln m}{\ln \ln m})$ gossip messages.*

[11] Since the gossip messages are dominant factor we ignore pair-wise messages sent for conflict resolution.

Proof Again, let Y be a random variable which represents the number of nodes that decide to become leaders at any slot. We want to find the maximum number of possible collisions in any slot. This problem is equivalent to *occupancy problem* or *balls in bin problem.* Using Chapter 3, Theorem 3.1 of [24], we know that with probability of at least $1 - \frac{1}{m}$, there are no more than $\frac{e \ln n}{\ln \ln m}$ coalition. Hence the proof. □

Theorem 2 *With a high probability, over all g groups the gossip messages complexity is $O(g\frac{\ln m}{\ln \ln m})$, for electing group leaders.*

Proof Follows by using union bound on lemma 5. □

This shows that the way SOG framework uses gossip communication for leader election keeps the communication overhead low.

So in essence analysis in this section we establish that the SOG framework has a low communication overhead and stabilizes quickly. This result demonstrates the scalability of the SOG framework which is further validated with the experimental observations.

6 Performance Evaluation

This section demonstrates how SOG achieves the scalability to a significant number of users and resources and excellent query performance when processing multi-attribute range queries. First we introduce Grid environments for the experiments. Then we study the effect of SFC attributes and measure query performance. We show that excellent performance is observed both in terms of the number of hops and query response time. Further we demonstrate the suitability of the SOG framework for routing multi-attribute range queries using Hilbert SFC. Finally we demonstrate that SOG approach is scalable to a large number of Grid nodes.

6.1 Modeling Grid Environments

For the experiments presented in this research, the following parameters are used to specify the Grid environments and model SOG on it.

1. Dimension of multi-attribute space represents the number of attributes that are considered for grouping. Experiments assume a 4-D, 6-D, 8-D, 10-D resource space[12].
2. Order of Hilbert space filling curve is used to control the granularity for assessing resource similarity. The lower the order of a Hilbert curve, the coarser its granularity is. We examine the effect of the order of Hilbert curve on query performance. The order values are set between 1 and 4. A curve of order 4 is quite sufficient for the purposes of grouping similar resources and keeping dissimilar resources apart.
3. Resource density reflects the percentage of nodes that have resources available. Since the resource information for these experiments was collected in near real-time from Open Science Grid (OSG) [26] nodes, we do not control the resource density for various queries. We would, however, study the effect of resource density on query performance.
4. Resource discovery query performance is measured using the average number of hops (taken by a query) as a metric. The number of hops is measured on SOG overlay network. Each node in the overlay generates an equal number of queries. Therefore, an originating node does not have any effect on the average hops observed in the experiments.

In our experiments we use the following types of range queries:

Q1: $1024 \leq$ freeMem[13] < 2048 && load1 $== 0$ && $2048 \leq$ totalMem < 3072 && procRun $== 0$
Q2: $1024 \leq$ freeMem < 3072 && load1 $== 0$ && $2048 \leq$ totalMem < 4096 && procRun ≤ 2
Q3: $1024 \leq$ freeMem < 3072 && load1 $== 0$ && totalMem ≥ 2048 && procRun ≤ 2
Q4: load1 $== 0$

From the queries listed above, we can see that query space increases from Q1 to Q4. In other words, we go from a more specialized query in Q1, which queries a small search space, to a more generalized query in Q4, corresponding to a large search space. Furthermore, Q4 represents a query search on only one attribute. The attributes that are not constrained are treated as "*don't care*" (i.e. any value for those attributes is acceptable). These queries are chosen because they are representative of various classes of queries. For each of the above query types, we conducted experiments with the order of Hilbert SFC ranging from 1 to 4.

[12] The resources considered for the experiments include: 1) free memory, 2) average 1 min system load, 3) total memory, 4) number of running processes, 5) user CPU usage, 6) total IO rate in, 7) total IO rate out, 8) disk free, 9) number of CPUs, 10) avg 5 min load. When considering a 4-D attribute space the first 4 listed resources are used, while for a 10-D attribute space all attributes are used.

[13] In Megabytes.

6.2 Experiment Setup

Resource usage information was collected from the Open Science Grid [26] using MonALISA [25] for this category of experiments.

6.2.1 Open Science Grid

The Open Science Grid (OSG) is a grassroots consortium of software,resource providers and researchers from universities, national laboratories and computing centers across the U.S. It provides a national production-quality Grid infrastructure by connecting distributed computing and storage resources via a common set of middleware to enable collaborative scientific discovery. This infrastructure is highly heterogeneous and consists of over 75 sites with thousands of computational resources and a large number of users. To simulate Grid environments, we use both static and dynamic resource information captured by near real-time monitoring of OSG resources.

6.2.2 MonALISA

MonALISA is agent-based dynamic distributed monitoring system. MonALISA collects static and dynamic availability information of OSG resources in near real-time, which allows us to emulate the OSG environment. The monitoring information that is collected includes information for nodes and clusters, network, jobs and services.

The experiments were conducted in a simulated environment that contains 1024 nodes. Experiments with 2048 and 4096 nodes were conducted in order to demonstrate scalability. Each participating node generates approximately 200 queries during the period of each measurement. Our experiments were run in a distributed environment using several 64-bit Athlon desktop machines.

6.3 Evaluating the Effect of Hilbert SFC Parameters

In this section we evaluate the effect of varying Hilbert SFC parameters of the SOG framework. Specifically we focus on the effect of changing the number of dimension and order of Hilbert SFC on performance.

Figure 5 illustrates the effect of order and number of dimension of Hilbert SFC on SOG performance, averaged over all types of queries, since the query type has no impact on the values being measured. An increase in the granularity of the Hilbert curve results in

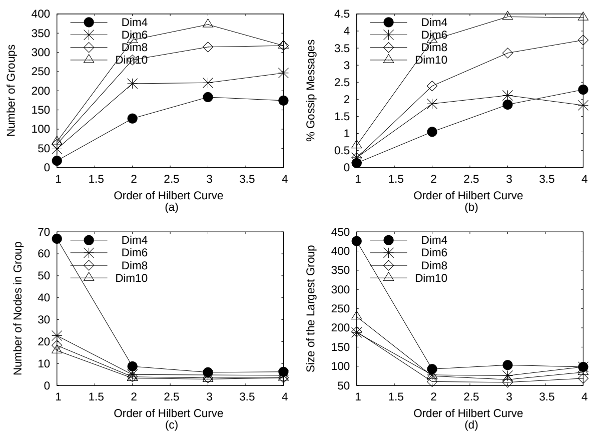

Fig. 5 Evaluating the effect of Hilbert SFC parameters on SOG Performance averaged over all queries (a) total number of groups, (b) gossip communication, (c) average group size, (d) maximum group size

an increase in the number of groups for every dimension. Furthermore, an increase in the number of dimension keeping order unchanged also causes an increase in the number of SOG groups (figure 5 (a)). This is because an increase in either order or dimension tends to make the groups more specialized, thereby groups are potentially split. A similar trend is observed with the percentage of gossip messages in figure 5 (b). The number of gossip messages needed for stabilization tends to increase with an increase in either of the Hilbert SFC parameter. This is a direct effect of an increase in the number of groups. Consequently, in figure 5 (c) we see a decrease in the average group size as well as the size of the largest group (figure 5 (d)) as either of the SFC parameter values increase.

The linear relationship between the amount of gossip message and the group size can be seen from figure 6. The observation from this figure is consistent with the theoretical analysis (Lemma 4) where we showed that the expected number of gossip messages needed to elect a group leader is constant.

From these experiments we can summarize the effect of Hilbert SFC parameter as follows. An increase in either the order or dimension of the SFC increases, tends to increase the number of resulting SOG groups. Consequently, this results in an increase in the number of gossip messages needed to self-organize and a decrease in the size of individual groups.

The findings above indicate that SFC attributes need to be chosen carefully. Having a high order or dimension will result in higher communication overhead because of the fragmentation of groups. On the other hand, low order and dimension values, will not

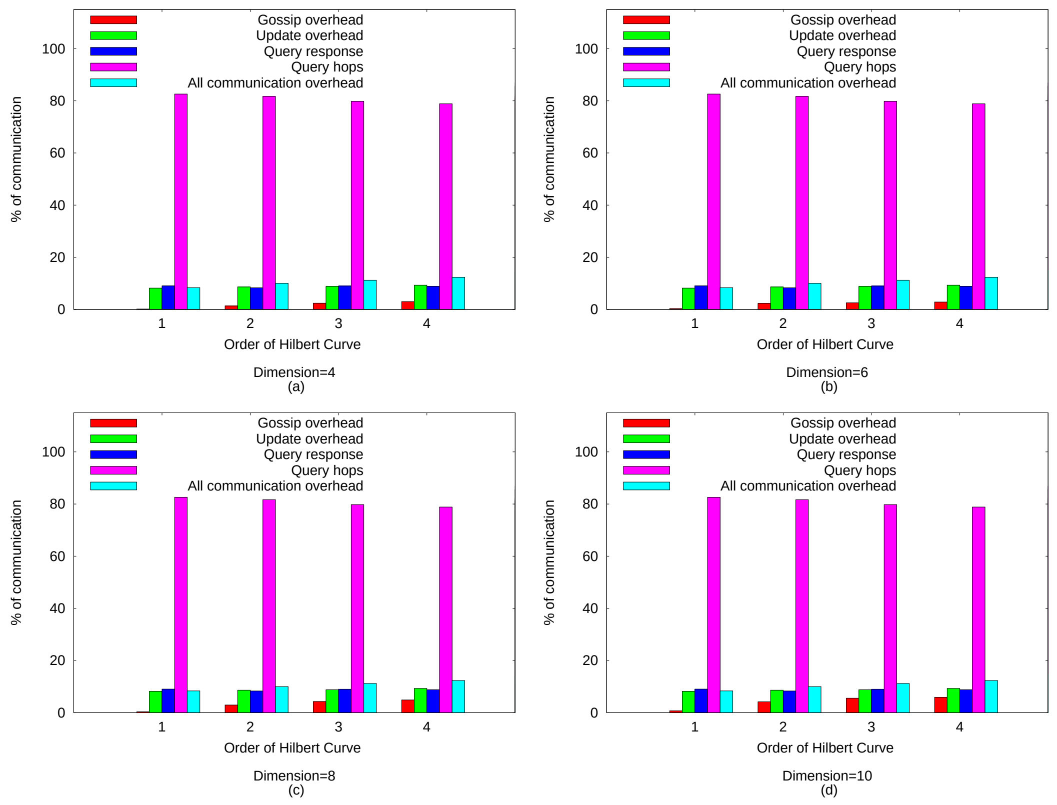

Fig. 7 Communication costs associated with SOG components. Observations from experiments conducted with dimension values (a) 4; (b) 6; (c) 8; and (d) 10.

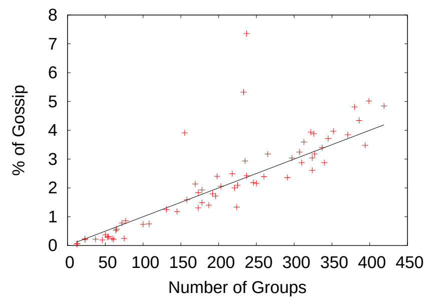

Fig. 6 Relation between number of groups and gossip communication

cluster resources into appropriate logical groups and will increase the possibility that some of the leader nodes would become hot-spots/bottlenecks during query processing. In other words, we need to balance these conflicting criteria when choosing the number of grouping attributes[14] and the order of the Hilbert SFC curve. Furthermore, as we shall see later in the experiments, the choice of the order and dimension also affects the performance of the SOG framework when handling resource discovery queries. The best choice should be determined by taking the nature of Grid environments and its expected usage into consideration.

6.4 Understanding Communication Costs

Since the SOG framework establishes a decentralized approach to resource discovery it is important to understand the communication overhead. Figure 7, illustrates the communication cost associated with various SOG components, by counting the messages exchanged over the lifetime of an experiment run. The figure highlights the communication costs incurred by different SOG components including cost of (1) gossip communication; (2) periodic updates from a worker

[14] The number of grouping attributes determines dimension.

node to its leader; (3) responses to query messages[15]; (4) hops taken by resource discovery queries; and (5) overall communication costs[16].

The results indicate that the communication overhead incurred by gossip is kept low (less than 5%), as is the cost for keeping the leader node updated with its worker node status (between 5 to 10 percent). So the overall overhead incurred for forming, maintaining and managing the SOG framework is kept quite low (between 5-15 percent). The remaining 85% of the time messages are exchanged in the SOG framework to handle resource discovery queries. Messages send trying to locate the requested resources account for around 70-80% messages exchanged, while approximately another 10-15% of the messages result from responses to the resource discovery queries. So the SOG framework spends a significant amount of its network bandwidth utilization handling resource discovery queries, achieving low communication overhead. The low communication overhead observed is a direct result of the careful design of SOG algorithms to minimize gossip communication. As we will see in the next section the average number of hops taken by resource discovery queries is quite low, hence the overall communication cost incurred by the SOG framework is low.

6.5 Measuring Query Performance

The query performance of the SOG framework is measured primarily in terms of the number of hops taken on the SOG overlay network. We also record the absolute time taken for each query and measure the reliability of our method.

Figures 8 shows the number of hops taken by resource discovery queries before they were satisfied. The results for each query type was averaged over multiple runs with different values of order and dimension of Hilbert SFC. Sub-figures (a) and (b) respectively illustrate the effect of order and dimension of Hilbert SFC on query performance. From these figures we observed that as the order increases the number of hops taken decreases. This is due to the fact that the groups are getting more specialized and queries are being directed more effectively to appropriate leader groups where resources can be found with an increased probability. At the first thought, one might expect the same kind of behavior where we increase the number of dimension, since this also makes the space more specialized. This

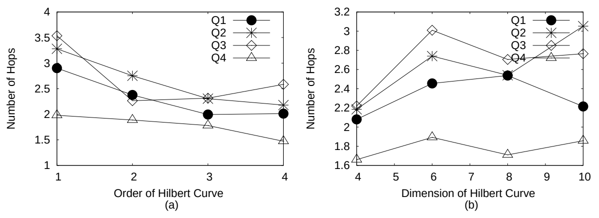

Fig. 8 Evaluating query performance using number of hops on SOG framework for queries as the (a) order of Hilbert SFC changes; (b) number of dimension used for constructing SOG groups changes.

would indeed be true if our queries were against the new attribute that were added for grouping. However, since queries Q1-Q4 are querying against attributes that were already used for grouping, the addition of new dimension actually causes some of the groups to fracture by specializing in these attributes which we were not interested in our queries. Hence, we see that the performance tends to slightly degrade as the number of dimension increases.

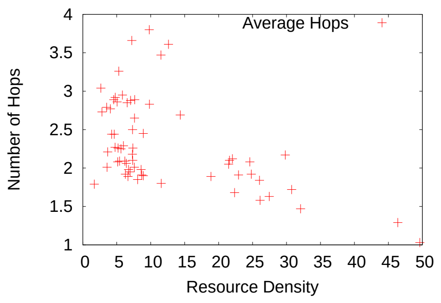

Fig. 9 Evaluating the effect of resource density on query performance

The effect of resource density on query performance is outlined in figure 9. Since there was no direct way to control or measure the density of dynamically changing resources, we measured it indirectly through the percentage of queries that were satisfied in zero hops. Because query generation is uniformly distributed, this measure gives us a reliable empirical measure of resource density. The graph indicates that the query performance remains relatively unaffected by the change in resource density. So essentially, query performance is equally good when the resources are either abundant

[15] A response message received back at the node originating the query.
[16] This includes the cost of gossip+update+other(e.g. messages exchanged when a node joins or leaves the system) messages.

or scarce[17]. Finding resources when they are scarce, usually requires a large number of hops and has traditionally been a problem with decentralized resource discovery methods. Due to its hybrid nature, the SOG framework is able to locate sparse resources quickly and is one of its major advantages. From figure 9, we also observed that the average number of hops is less than 4 in all the experiments we conducted. From these observations it is evident that the SOG framework achieves excellent query performance, based on the number of hops measured on the overlay network.

Table 1 answers the question: What is the average response time spent by any query from the time it is initiated to the time a resource discovery is made? The response time is measured in seconds. These numbers provide an estimate of how long it takes to identify a resource. It is observed that the time taken to process resource discovery queries ranges between 1.89 sec and 6.56 sec. The processing time is found to be relatively independent of SFC parameters of order and dimension as well as query type. It is important to note that there are some factors, which might affect query performance in a real world Grid deployment, may not be captured in the experimental settings. In particular, we have identified two factors: 1) nodes were being simulated within the university network. This may cause some underestimation of the average times. A network latency correction factor (which is a function on the WAN latency and average number of hops taken) needs to be added when dealing with in a WAN environment; 2) multiple simulated nodes are running on the same physical CPU and hence there might be contention for CPU resource usage which might cause overestimation of query processing times.

Since SOG does not provide any guarantee on discovering resources, we measure reliability as a final factor for evaluating query performance. Reliability is measured as the probability that the SOG framework successfully locates the resource. A very high reliability rate (over 99.9%) was observed during each of the experiment runs (figure 10), irrespective of the SFC parameters or query type. So if the requested resource is available, there is a very high probability that the SOG framework will successfully locate the resource.

The results in this section indicate that SOG achieves a highly reliable query performance measured both in number of hops and average time taken.

[17] It gets better when the resources are very abundant. For e.g. when every other node has the requested resource (resource density 50%) the average hop drops to 1.

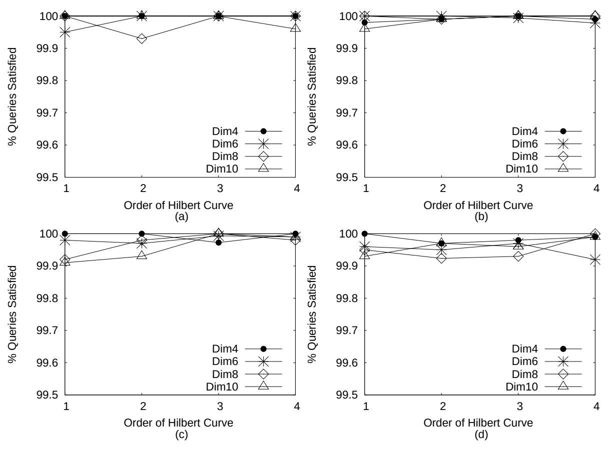

Fig. 10 Reliability of query processing on SOG framework for queries (a) Q1, (b) Q2, (c) Q3, (d) Q4

6.6 Scalability

In this section we demonstrate the scalability of SOG with the number of nodes. For this purpose we conduct experiments with 1024, 2048, and 4094 nodes, with 4 and 10 dimensional attribute space. As in previous experiments, we measure the effectiveness of the grouping scheme, communication overhead and query performance. The experiment runs associated with the results in this section, use queries Q1 and Q4 and average out the results.

6.6.1 Measuring Effectiveness of the Grouping Scheme

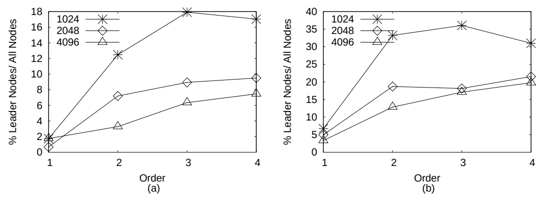

Fig. 11 Evaluating the effect of number of nodes on self-organized grouping, with (a) *4*-, (b) *10*-, dimensions.

We observed that with other factors (order and dimension) unchanged, as the number of nodes in the system increases, the percentage of leader nodes in the system decreases (figure 11). For e.g. with dimension=4 and order=2; the percentage of nodes that are leaders is (a) 12.48% (results in 128 groups); and (b) 3.28% (corresponds to 135 groups); in an experiment conducted with 1024 and 4096 nodes respectively. So even though

Table 1 Average query processing time (in sec) for resource discovery queries

	Q1				**Q2**				**Q3**				**Q4**			
Order → **Dimension ↓**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
4	2.73	2.38	2.73	2.62	2.90	2.81	2.78	2.59	3.62	2.48	2.50	2.25	1.89	2.26	2.13	2.13
6	3.74	3.34	2.78	2.79	3.04	3.40	4.33	6.56	4.06	3.06	4.05	3.19	2.30	2.22	1.97	2.63
8	3.14	3.87	2.75	2.82	3.27	3.62	2.72	3.17	3.62	2.78	2.72	4.14	2.44	2.34	2.57	1.68
10	3.26	3.46	3.43	2.80	5.85	3.90	3.08	2.85	3.58	4.40	3.45	4.30	2.53	2.66	2.48	2.15

increasing the number of nodes results in a slight increase in the absolute number of groups, the number of groups per node tends to go down. This observation clearly demonstrates our grouping premise that nodes tend to cluster together into groups and serves as an important observation to demonstrate the scalability of the SOG framework.

6.6.2 Communication Overhead

As we already saw gossip communication and periodic updates published from a worker node to its leader are the two major components of the communication overhead incurred by SOG. The graphs from figure 12 demonstrate the effect of increasing the number of nodes on gossip communication overhead.

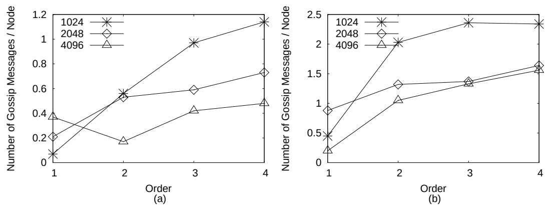

Fig. 12 Plotting the number of gossip communication per node, with (a) *4*-, (b) *10*-, dimensions.

We observed that the share of the number of gossip messages per node decreases with an increase in the number of nodes (figure 12). This is due to the fact that as the number of nodes increases, only a small number of new groups are created (as in the previous section). Also with an increase in the number of nodes we expect to service an increased number of queries, this would actually decrease the percentage of gossip messages as a percentage of total messages. As with the overhead resulting from the updates being sent from worker to leader, the number of messages will increase linearly with the number of nodes, but the number of queries serviced can also be expected to increase linearly. So as a percentage of the total communication costs, the cost of update messages tend to stay constant. The overall communication overhead, as a percentage of total communication, in these experiments tends to drop. This phenomenon positively impacts the scalability of the SOG framework.

6.6.3 Query Performance

Now that we have examined the effect of increase in the number of nodes on grouping performance and communication overhead, we evaluate its effect on query performance.

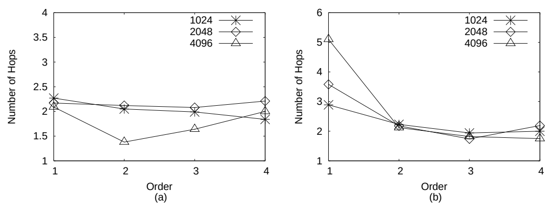

Fig. 13 Plotting the average number of hops taken to satisfy queries, in presence of 1024, 2048 and 4096 system nodes, with (a) *4*-, (b) *10*-, dimensions.

Figure 13 plots illustrate the average number of hops. The only noticeable effect, consistent with the previously discussed observation in section 6.5, is that an increase in order causes potential improvement in query performance, while an increase in dimension might potentially have a negative impact on performance. These figures show that query performance is largely unaffected by the number of nodes in the system. This is due to the fact that query performance depends on the effectiveness of the self-organized grouping scheme rather than the number of nodes in the system.

As the number of nodes increases, the grouping scheme proposed by SOG, tends to decrease the percentage of groups per node, as well as the communication overhead without any impact on query performance. Hence the SOG framework scales effectively to a large number of nodes.

In summary, the experiments demonstrate the feasibility of using the SOG framework to handle various types of resource discovery requests. Hilbert space-filling curves provide an effective way to tune the grouping mechanism within a Grid environment. The communication overhead associated with the SOG framework is observed to be low. The number of hops taken by a query on the SOG framework is low and the response to a resource discovery query is quick. SOG achieves superior resource discovery performance while being affected little by resource density, resource type, query search space, or system size.

7 Conclusion and Future Work

Below we summarize the major contributions of this paper.

– This paper introduces a self-organization framework for Grid resource discovery based on grouping together similar resources.
– The hybrid solution characterized by SOG takes advantages of both centralized and decentralized approaches, while avoiding their pitfalls.
– The SOG framework is developed specifically to handle dynamically changing resources.
– Integrating space-filling curves within the SOG framework, this paper illustrates a method to handle multi-attribute range queries.
– The proposed solution provides superior lookup performance while maintaining a high level of scalability with minimum overhead.

While the paper focuses on resource discovery in Grid environments, the approach is generically applicable to peer-to-peer environments.

The experiments demonstrated that SOG achieves its design goals. More specifically, it is observed that the handling of each query using SOG takes on average a small number of hops as resource densities and system sizes are varied within wide ranges in the experiments. Experiments also illustrate the suitability of using Hilbert space-filling curve with the SOG framework to achieve efficient Grid resource discovery for multi-attribute range queries. Observations from experiments indicate that a change in the density of a resource being discovered has little effect on query performance. In contrast to other existing decentralized approaches, SOG performs well when the resource are very sparsely available. Furthermore, the performance remains largely unaffected by different query types, with varying sizes of query space and the sizes of Grid environments. The mathematical analysis proves that the SOG framework stabilizes quickly with low overhead and validates experimental observations. Overall, the SOG approach is independent of the nature of resources being discovered, query types and Grid environments, and thus, provides an scalable framework for resource discovery.

In SOG's implementation of Hilbert SFC, the number of groups (or the maximum group size), is controlled by changing the order of Hilbert SFC. A potential problem of merely increasing the order is that it would breakup all groups irrespective of their sizes. A better approach is to have a targeted higher order curve in the regions of the resource space that are densely populated. Conversely, sparsely populated regions would have a lower order curve. This approach is adaptive to group size, and consequently reduce the risk of developing hot-spots while keeping communication overhead minimal. It would also be an interesting research topic to study the effect of adaptive groups on query performance and communication overhead.

Another related aspect to investigate is what would be the effect of creating a hierarchical structure of leaders. This would imply that initially we will all begin with an order 1 Hilbert SFC and elect top level leaders based on this curve. Only these top level leaders will be known to the entire system. In each of these groups we can have an order 2 Hilbert SFC curve to elect leaders for each sub-group that will only be known to the members of the group, and so on till we get sufficient partitions of resources. This will have the advantages of adaptive grouping schemes, while having an ordered structure and lowering the communication overhead. But using this approach query performance might be degraded, because the diameter of the graph is increased and the top level leaders might be overloaded. It is an interesting research topic to explore the exact nature of this trade-off and study the effectiveness of such an approach.

Acknowledgements The authors will like to thank the Open Science Grid (OSG) for the data that was used to support the computational experiments. We are also greatful to the anonymous reviewers for their suggestions to improve the paper.

References

1. A. Andrzejak and Z. Xu. Scalable, Efficient Range Queries for Grid Information Services. In *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*, page 33, Washington, DC, USA, 2002. IEEE Computer Society.
2. T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer. Space-filling curves and their use in the design of geometric data structures. *Theoretical Computer Science*, 181(1):3–15, 1997.
3. G. Ayalvadi, A. Kermarrec, and L. Massoulie. SCAMP Peer-to-peer lightweight membership service for large-scale group communication. *In Proc. 3rd Intnl. Wshop Networked*

Group Communication (NGC '01), LNCS 2233, Springer:44–55, 2001.

4. A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-attribute Range Queries. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 353–366, 2004.
5. A. R. Butz. Alternative Algorithm for Hilbert's Space-filling Curve. *IEEE Transaction on Computers*, C-20:424–426, Apr 1971.
6. E. Caron, F. Desprez, and C. Tedeschi. Enhancing computational grids with peer-to-peer technology for large scale service discovery. *Journal of Grid Computing*, 5(3):337–360, September 2007.
7. G. Chen, C. Ping Low, and Z. Yang. Enhancing Search Performance in Unstructured P2P Networks Based on Users' Common Interest. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):821–836, 2008.
8. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. *In Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, 2001.
9. S. Eva, R. Manohar, and M. Chandy. An Analysis of Leader Election for Multicast Groups. Technical report, AT&T Labs-Research, Feb 2002.
10. A. Forestiero, C. Mastroianni, and G. Spezzano. Building a peer-to-peer information system in grids via self-organizing agents. *Journal of Grid Computing*, 2008.
11. A. Forestiero, C. Mastroianni, and G. Spezzano. So-grid: A self-organizing grid featuring bio-inspired algorithms. *ACM Trans. Auton. Adapt. Syst.*, 3(2):1–37, 2008.
12. I. Foster and A. Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. pages 118–128. 2003.
13. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the Grid: An open grid services architecture for distributed systems integration. Technical report, Globus Project, 2002.
14. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 2001.
15. A. Iamnitchi and I. Foster. On Fully Decentralized Resource Discovery in Grid Environments. In *GRID '01: Proceedings of the Second International Workshop on Grid Computing*, pages 51–62, London, UK, 2001. Springer-Verlag.
16. G. Jin and J. Mellor-Crummey. SFCGen: A framework for efficient generation of multi-dimensional space-filling curves by recursion. *ACM Trans. Math. Softw.*, 31(1):120–148, 2005.
17. A. Kermarrec, L. Massouli, and A. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):248–258, 2003.
18. G. Koloniari and E. Pitoura. Bloom-Based Filters for Hierarchical Data. *In 5th Workshop on Distributed Data and Structures*, 2003.
19. J. Lawder. Calculation of Mappings between One and n-dimensional Values Using the Hilbert Space-Filling Curve, 2000.
20. M. Li, W. Lee, A. Sivasubramaniam, and J. Zhao. SSW: A Small-World-Based Overlay for Peer-to-Peer Search. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):735–749, 2008.
21. C. Mastroianni, D. Talia, and O. Verta. A super-peer model for resource discovery services in large-scale grids. *Future Generation Computer Systems*, 21(8):1235–1248, 2005.
22. P. Merz and K. Gorunova. Fault-tolerant Resource Discovery in Peer-to-peer Grids. *Journal of Grid Computing*, 5(3):319–335, September 2007.
23. B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.
24. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, NY, USA, 1995.
25. H.B. Newman, I.C. Legrand, P.Galvez, R. Voicu, and C. Cirstoiu. MonALISA: A Distributed Monitoring Service Architecture. In *CHEP*, La Jola, California, March 2003.
26. Open Science Grid (OSG). http://opensciencegrid.org, 2008.
27. D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. In *HPDC*, Research Triangle Park, NC, July 2005.
28. A. Padmanabhan, S. Wang, S. Ghosh, and R. Briggs. A Self-Organized Grouping (SOG) Method for Efficient Grid Resource Discovery . In *GRID 2005*, Nov 2005.
29. S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Prefix Hash Tree. In *PODC*, 2004.
30. R. Raman, M. Livny, and M. Solomon. Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching. In *Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing (HPDC-12)*, Seattle, WA, June 2003.
31. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM*, 2001.
32. RFC-Gnutella. The Gnutella Protocol Specification v0.4, 2004.
33. C. Schmidt and M. Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, Washington, DC, USA, 2003. IEEE Computer Society.
34. S. Schulz, W. Blochinger, and H. Hannak. Capability-Aware Information Aggregation in Peer-to-Peer Grids. *Journal of Grid Computing*, 7(2):135–167, June 2009.
35. D. Spence and T. Harris. XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, Washington, DC, USA, 2003. IEEE Computer Society.
36. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
37. A. C. Viana, M. D. de Amorim, Y. Viniotis, S. Fdida, and J. F. de Rezende. Twins: A Dual Addressing Space Representation for Self-Organizing Networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(12):1468–1481, 2006.
38. B. Yang and H. Garcia-Molina. Designing a super-peer network. In *IEEE International Conference on Data Engineering, (ICDE 2003)*, March 2003.
39. Cheng Zhu, Zhong Liu, Weiming Zhang, Weidong Xiao, Zhenning Xu, and Dongsheng Yang. Decentralized grid resource discovery based on resource information community. *Journal of Grid Computing*, 2(3):261–277, September 2004.