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

Anand Padmanabhan*‡, Shaowen Wang*[1] Sukumar Ghosh‡, and Ransom Briggs*‡
*Grid Research and educatiOn group @ IoWa (GROW), ITS-Academic Technologies, The University of Iowa
‡Deptartment of Computer Science, The University of Iowa
{anand-padmanabhan-1,shaowen-wang,sukumar-ghosh,ransom-briggs}@uiowa.edu

Abstract— **This paper presents a self-organized grouping (SOG) method that achieves efficient Grid resource discovery by forming and maintaining autonomous resource groups. Each group dynamically aggregates a set of resources that are similar to each other in some pre-specified resource characteristic. The SOG method takes advantage of the strengths of both centralized and decentralized approaches that were previously developed for Grid/P2P resource discovery. The design of the SOG method minimizes the overhead incurred in forming and maintaining groups and maximizes resource discovery performance. The way SOG method handles resource discovery queries is metaphorically similar to searching for a word in an English dictionary by identifying its alphabetical groups at the first place. It is shown from a series of computational experiments that SOG method achieves more stable (i.e., independent of the factors such as resource densities, and Grid sizes) and efficient lookup performance than other existing approaches.**

Keywords: Grid resource discovery, peer-to-peer, self-organized grouping

I. Introduction

Grid computing enables users to assemble large-scale geographically distributed computational resources to create a secure virtual supercomputer that can be used to accomplish a specific computational goal [1]. This assemblage of distributed resources is dynamically orchestrated to support coordinated resource sharing through Grid middleware. At present, however, the Grid represents an extremely complex distributed computing environment for developing applications because:

1) Grid resources (e.g., CPU, network, storage, and special-purpose instrument) are heterogeneous;
2) Grid resources are dynamic, and they tend to have faults that may not be predictable; and
3) Grids are often distributed across security domains with large number of resources involved.

Grid resource discovery refers to the process of locating satisfactory resources based on user requests [2]. This process represents an important step based on which resource reservation and task scheduling can take place to enable Grid application development. For illustrative purposes we would consider a resource to be a dynamic characteristic of a computing element (e.g., CPU load and available memory). A node is an element which contains a combination of these resources. So resource discovery would be to find a node that would satisfy all resource requirements specified in a resource request. Due to the complexities inherent to a Grid environment, it is challenging to develop efficient methods to discover dynamic Grid resources. Grid resource discovery methods must intelligently handle resource requests by searching a small subset out of a large number of accessible resources, the status and availability of which dynamically change.

The purpose of this paper is to demonstrate a self-organized grouping (SOG) method for efficient Grid resource discovery. A group is formed by a collection of nodes with some similarities in their characteristics. Each group is managed by a leader, which is the group representative and consists of members that serve as workers. Resource requests are ideally forwarded to an appropriate group leader that would then direct it to one of its workers, thereby achieving good performance. The SOG method develops the following three strategies:

1) Control resource heterogeneity to the extent that all the nodes in a group have some statistically-similar resource characteristic during a certain time period;
2) Capture resource dynamics through grouping around dynamically elected leaders as well as publishing worker resource information [3] to the leaders; and
3) Enable the assemblage of large number of resources for applications within and across VO boundaries through a limited number of dynamic groups that are potentially searched faster than directly searching them individually.

The main thesis of this research is that the SOG method achieves efficient Grid resource discovery through these strategies. These strategies are implemented to classify nodes into a number of groups based on the choice of specific similarity characteristics and the use of appropriate statistical measures (e.g., weighted temporal means of resource readings).

Initially, the SOG method treats Grids as an unstructured peer-to-peer network of available nodes (e.g., Gnutella [4]) to avoid the unique ID and scalability problems that would exist for structured peer-to-peer networks such as CHORD [5]. A light-weight gossip mechanism [6] is used to communicate statistical summary information of a resource characteristic to facilitate leader election in dynamic groups. Resource discovery using the SOG method takes advantage of the capabilities of both centralized and decentralized approaches. Decentralized capabilities support forwarding queries based on resource grouping. Centralized capabilities allow the search for

[1]Corresponding Author

requested resources within a group context. This hybrid strategy of combining centralized and decentralized capabilities in a self-organized manner is designed to achieve better lookup performance than decentralized approaches while providing better scalability than a centralized approach.

The overhead of forming and maintaining groups is kept low so as not to offset the potential performance gain of the hybrid strategy. Special attention is paid to the network bandwidth consumed by the gossip protocol. Simulation-based experiments were conducted to compare query performance among the cases that have different densities and types of resources. Our performance study compares the SOG method with other selected resource discovery methods. These comparisons validate the thesis of this research.

The rest of the paper is organized as follows. Section 2 illustrates the design of the SOG method by articulating the properties of self-organized groups and the functions of several algorithms that implement SOG. Section 3 describes the details of these algorithms. Section 4 presents a simulation-based comparative study of the SOG method. Section 5 relates the SOG method to other Grid resource discovery research. Section 6 concludes this paper by highlighting the major findings of this research, and providing pointers to future research.

II. SOG Design

In this section we briefly discuss the SOG overlay network, followed by an overview of SOG components and their functions.

A. Overlay Network

In order to form and maintain groups as well as process queries, we implement a two-layer overlay network. The lower layer is formed using a lightweight gossip protocol [6]. In this layer a node has $O(\log n)$ neighbors [6]. This layer is used to forward gossip messages (for leader election) and some query messages (as a last resort). A query is randomly forwarded using the lower layer, when upper layer fails to find a node to forward query.

The upper layer reflects the interconnection among leaders and workers. It is composed 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. Figure 1 shows a visual representation of the upper-layer. One extreme case occurs when there is a single group, resulting in a star graph, that is equivalent to a single/centralized server architecture. The other extreme case takes place when each node is a group in itself, leading to a completely connected graph. 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 discussed above. The SOG method is initialized by specifying a similarity threshold to control the number of groups. The tradeoff that guides the threshold specification is between communication overhead and load on group leaders. If there are few groups, then the leader nodes of these groups might be forced to handle an excessive amount of load and thus become performance bottlenecks; while if there are too many groups, then the cost of communication required to form and maintain these groups would be significant.

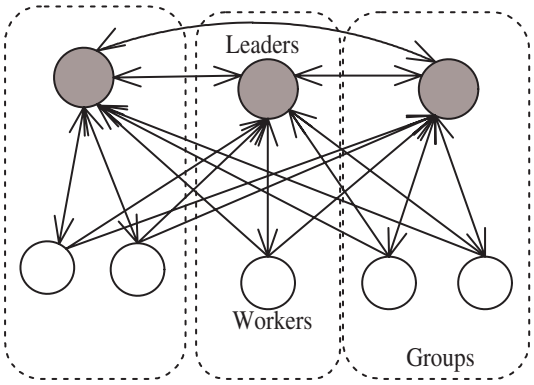

Fig. 1. The upper level graph maintained by SOG

B. Overview of SOG Components

Self-organized groups are formed in a bottom-up fashion, i.e., each node starts as a group by itself. Through execution of SOG components, groups merge into larger groups based on the similarity threshold currently specified. This paper does not specifically address the evaluation of similarity threshold. Instead, our focus is placed on the development and evaluation of self-organized grouping capability. This capability is implemented by the following algorithmic components that are executed on each node in a nondeterministic way:

1) Publish resource information;
2) Handle group forming and maintenance;
3) Receive and process system messages (like *subscribe, proclaim leader, resolve conflict*);
4) Receive and process query messages;

The first component publishes current resource information periodically from a node to its own leader. This is the only active publishing that occurs in SOG, thereby enhancing the scalability of deploying SOG. This information is also made available to query processing.

The functions of creating, checking, and maintaining groups are handled by the second SOG component. Group checking takes place periodically to ensure that each node belongs to an appropriate group as a selected resource characteristic dynamically changes. When a new node that joined the Grid has sufficient temporal samples, it calculates the statistical characteristic. It then searches for, and if successful joins an existing group, the statistical characteristic of which is similar to that of the joining node. 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 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 hops made by queries. The information about group leaders and their characteristics is made available for query processing through the third SOG component, which handles incoming gossip messages. This availability helps directly forward a query to an appropriate leader that is in charge of a group with resources similar to the ones requested by the query (with respect to selected similarity characteristic). The leader is then able to direct the query to one of its workers that is able to satisfy the query.

The SOG querying mechanism limits a query's scope from the entire Grid to a group location with 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 is sufficiently flexible to forward the query to another group leader that may satisfy the query. This forwarding mechanism between group leaders achieves high resource discovery performance by keeping resource discovery scope at group level. In case no leader can be identified, queries are randomly forwarded to a gossip neighbor. The last component of the SOG algorithms handles query processing.

To achieve a small number of gossip messages while actively maintaining the groups formed and efficiently handling resource discovery queries, the design of SOG algorithms treats group leaders differently than other resources. More specifically, new replacement group leaders are introduced to maintain the major information for the existing groups, the leaders of which are leaving. This strategy avoids dissolution of a group into many new groups of individual resources in this leader-leaving situation, and thus saves gossip messages that would be required without the application of the strategy.

In addition, since leader election is based on the evaluation of their local information, multiple nodes may contend to become leaders though they are statistically similar. The detection and resolution of such a conflict condition are handled by the third component.

In summary self-organized groups are designed to have the following three characteristics.

1) Each group includes a collection of nodes with a single representative node called a leader.
2) The nodes within each group are statistically similar in a selected resource characteristic. Similarity assessment involves the comparison of temporal samples of a selected characteristic among nodes. For example, t-test can be used for such comparison of means. The selection of a specific resource characteristic is guided by understanding common resource requirements from applications that issue resource discovery requests.
3) The size of each group is indirectly determined by a similarity threshold that is chosen based on the number of Grid resources, resource density, and the load of resource discovery requests. This threshold specifies a maximum range within which a resource characteristic is deemed similar. There will be a small number of nodes in a group and the number of groups will be relatively large should a threshold be held tight.

III. SOG Algorithm Components

This section presents the data structures and algorithm components that deal with group forming and maintenance, node-subscribe/unsubscribe as well as query processing.

A. Data and Query Model

In SOG, 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*** [2] in 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 $<$, $>$, $\leq$, $\geq$, $=$, $\neq$.

B. 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: $isLeader$; {Tells if a node is a leader}. Initially $isLeader \leftarrow false$
7: x; {Rounds after which the statistic is recalculated}. Initially $x \leftarrow pre-defined\ constant$

On leader nodes additionally 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 $V \leftarrow \emptyset$

C. 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 [6]. 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.

A non-leader node unsubscribes in a manner similar to that depicted in [6]. However 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.

[2] We plan to add ***char, string*** types in next version.

Algorithm 1 Group Formation, Maintenance: at node n

$event\_count \leftarrow 0$ {Timer count for periodic recalculation of statistic (c)}

1: r ← 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 ← recalculate_c(R);{Recalculate the temporal characteristics (c) given a set of resource values (R)}
6: event_count ← 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: gossip_proclaim_leadership(n,c);{Proclaim node as leader through gossip}
13: **else**
14: {A new leader was found}
15: Inform node l' about n joining its group
16: Inform node l about n leaving its group if $l \neq undefined$
17: $l \leftarrow l'$ {Assign new leader}
18: **end if**
19: **else if** $n.isLeader \wedge not\_similar(c, S(n)) \wedge change\_persistent()$ **then**
20: {Node n is a leader that has moved away from its group range and the move appears persistent}
21: replace_leader();
22: **end if**
23: **end if**

D. Group Formation and Maintenance

Algorithm 1 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 temporal usage/availability characteristic and the statistical similarity measure used (steps 5-7). If not, it tries to search 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 a node is a leader then it will first find a replacement leader before it leaves the group, so as to maintain grouping information (step 21).

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 this strategy, 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 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. Once a replacement leader has been found, the current leader identifies an appropriate group for itself.

A conflict occurs when, due to autonomous decisions made at node level, multiple nodes attempt to form different groups that have statistically similar characteristic. The detection of the conflict occurs on at-least one of the conflicting leaders. The multiple groups then merge, and the leader of the larger group becomes the leader of the merged group; ties are broken randomly. If there are conflicts among more than two groups, they are handled in pairs, and our conflict resolving algorithm guarantees that exactly one leader is elected for the merged group after all conflicts are resolved.

E. Query Routing

A user will submit a query to a local node, the system will return to them 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 leader's current knowledge, the worker node would be 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. If such a leader is not available, then the query is randomly forwarded.

F. Handling Multiple Attributes

The SOG algorithms are suitable for both single- and multi-attribute queries. The main difference between handling a single- and multi-attribute query is that we need to make the selection of a specific resource characteristic for grouping in the multi-attribute case. Selecting an appropriate resource characteristic depends on the characteristics of Grid environments as well as application requirements, and it can have an impact on query performance. This single characteristic may simply be one of the resource attributes or a function based on some or all of the attributes involved. For evaluation purposes, this research adopts single resource attribute selection. One possible function that could be used is based on space-filling curves [7], that will map an n-dimensional space into a 1-d space. We could use this 1-d attribute as the grouping characteristic. Squid [8] uses a similar idea to search for keywords in static multi-attribute data. More work needs to be done to explore the suitability of this idea.

Another way of addressing multi-attribute queries is to repeat grouping process on multiple attributes. Then when a query arrives, based on the selectivity of the query it could be routed using the most selective attribute. Mercury [9] uses this method to route queries, though they deal with static data

elements. This method may provide good performance for all the attributes on which the groups are formed. But the drawback is that the number of gossip messages will increase linearly with the number of grouping attributes. In addition, resource information would need to be replicated once for each attribute. Further research needs to be conducted to assess the cost versus benefit of using this method.

IV. Experimental Evaluation

A set of computational experiments were designed to study SOG's performance. These experiments are used to observe the effect of resource density and system size.

A. Grid Environment Modeling

We use the following parameters to specify our Grid environment.

1) Resource density reflects the abundance/scarcity of resources. It represents the percentage of nodes that have resources.
2) Resource type: We define two types of dynamic resources: binary-valued and continuous-valued. Binary-valued resources cannot be shared and satisfy a single request at a time, i.e. a resource is either available or unavailable. Continuous-valued resources on the other hand can be partially used and are able to satisfy multiple requests at the same time (e.g. memory).
3) User queries: We use average number of hops (taken by a query before it is satisfied) as a metric to evaluate SOG's performance. The number of hops is measured on SOG overlay network. Each node has an equal probability to generate queries.
4) Statistical characteristic and similarity measure affect the number of groups formed as well as the size of each group. We used weighted temporal mean of the selected resource characteristic as the statistical characteristic, and a range around the mean for the similarity measure.
5) Resource distribution describes how resources are distributed among nodes. In our experiments we assumed a random distribution.

B. Results

Experiments were conducted on a IA-32 Linux cluster with 11 physical nodes, each having a shared memory dual-CPU.

Our Grid environment has 1280, 1024, 256 simulated nodes in different runs. Each of these simulated nodes include at most one requested resource. The number of nodes containing the requested resource was varied by changing resource densities from $\frac{1}{4}$ to $\frac{1}{256}$. No limit placed on the number of times a query gets forwarded, i.e. no external hop-limit was imposed. There was also no limit placed on group sizes. An average of 15000 queries were handled for each simulation run.

Binary-valued resource type was used for the first subset of experiments in which the system converges, in a high percentage of cases, to two groups. This case is Condor-like [10] in a sense that the leader for the group that has resources is equivalent to a matchmaker. Continuous-valued resources were used for the second subset of experiments. It was observed that the system converged to 10-15 groups.

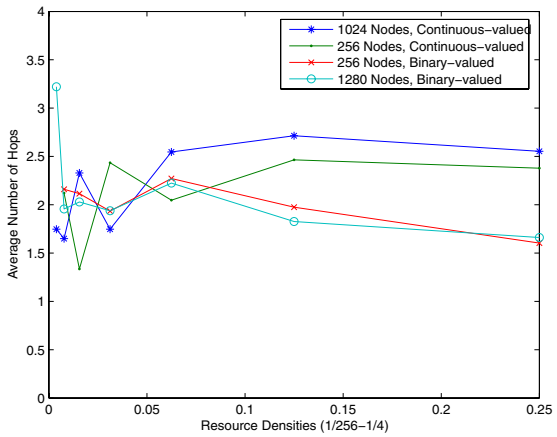

Fig. 2. The relationships between the average of number hops taken by each query and resource density

Figure 2 shows a plot of average number of hops taken by queries at various resource densities and system sizes. It was observed that the average number of hops stay at around 2, irrespective of the variance in resource density, resource type and system size. This observation indicates that most queries are dispatched to an appropriate leader, which is able to find a suitable node to satisfy them. This also demonstrates that query handling performance in SOG mechanism is largely unaffected by resource density, resource type, or system size.

The results in Figure 2 demonstrate that the SOG method achieves better stability and efficiency in query handling than [11], [12] because in [12] there is an increasing trend in average number of hops as resource density decreases while in [11], there is an increasing trend in average number of hops as system size increases. Due to space constraints, multi-attribute experiments are not presented here. We are working on more multi-attribute experiments and larger scale simulations to verify scalability, stability and efficiency of the SOG method.

V. Related Work

Current solutions to Grid resource discovery fall broadly into two categories, centralized and decentralized. Centralized systems are better in terms of lookup performance, but they scale poorly. Decentralized systems are generally scalable in terms of the number of nodes that can be handled, but a significant deterioration in average lookup performance is observed. Decentralized peer-to-peer technologies, used extensively for file sharing, are divided into two major categories: unstructured and structured. Unstructured peer-to-peer networks like Gnutella [4], that service requests using broadcasts, generally utilizes significant bandwidth and does not provide guarantees to find existing resources. Structured peer-to-peer networks like CHORD [5], generally guarantee finding existing resources within a finite number of hops, but the prerequisite is that every resource-value pair has to

have a unique ID. This limits the scalability of Grid resource discovery.

Condor Matchmaker [10] uses a centralized architecture to discover computational resources based on user specifications. Matchmaker is a central server responsible for matching the ads between resource consumers and resource providers. Globus MDS-2 [13] uses a distributed architecture to provide resource information services. A resource provider can use a registration protocol to register resource information to GIIS (Grid Index Information Service) [13], and a user can use a query protocol to access resource information from GIIS and GRIS (Grid Resource Information Service) [13]. GIIS and GRIS adopt a hierarchical approach, with GIIS gathering information from multiple GRIS servers. Li, et.al, [12], present a SD-RT (Shortest Distance Routing-Transferring) algorithm formulation based on their *routing-transferring model* to solve the resource discovery problem. Their main idea is to have a resource router which will be able to direct queries to resource providers, using routing tables. Different request forwarding schemes including, random, experience-based + random, best-neighbor + random, best-neighbor + experience-based are discussed and evaluated in [2].

Work done in the context of file and peer location in distributed P2P system are also relevant to our study. Harchol-Balter, Leighton, and Lewin [14] were the first to address and analyze the peer-location problem in a dynamic decentralized system and presented a new Name-Dropper algorithm. Kutten and Peleg in [15], introduced an asynchronous algorithm for peer discovery in a weakly-connected network, while Abraham and Dolev [16] addressed dynamic peer joins and leaves. These peer-location algorithms concentrate on locating all peers from any node. The problem we are addressing is different from the peer-location problem in that we aim to find an existing resource with a minimum number of hops.

VI. Conclusion and Future Work

In this paper, we introduce a self-organized grouping method for efficient Grid resource discovery. The SOG method is designed to significantly improve resource discovery lookup performance while maintaining a high level of scalability with minimum overhead. Our experiments demonstrated that the SOG method achieves its design goals. More specifically, we observed that each query handled based on the SOG method takes 2 to 3 hops on average. This observation was consistent when resource densities and system sizes were varied within wide ranges in the experiments. This consistency indicates that resource density and system size have little impact on query performance in the SOG method. Larger scale experiments are being conducted to further verify this consistence.

In the current SOG implementation, there is no direct way to control the size of a group. We are investigating a explicit way of controlling group sizes by specifying two new parameters: merge-size and split-size. This will cause large groups ($>$splitsize) to split and small groups ($<$mergesize) to merge. It is an interesting research topic to assess the feasibility of developing SOG grouping schemes by taking advantage of the capabilities of handling multi-attribute queries provided by [7], [9]. This type of research may help design flexible grouping schemes that are adaptive to system size, query load and resource availability.

Acknowledgment

This research is partially supported by the HawkGrid project funded by the Office of the Vice President for Research at The University of Iowa. Computational resources used for experiments include a cluster supported by the U.S. NSF iVDGL (international Virtual Data Grid Laboratory) project.

References

[1] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, 2001.

[2] 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*. London, UK: Springer-Verlag, 2001, pp. 51–62.

[3] S. Wang, A. Padmanabhan, Y. Liu, R. Briggs, J. Ni, T. He, B. Knosp, and Y. Onel, "A Multi-Agent System Architecture for End-User Level Grid Monitoring Using Geographic Information Systems (MAGGIS): Architecture and Implementation," *Lecture Notes in Computer Science*, vol. 3032, pp. 536–543, 2004.

[4] RFC-Gnutella, "The Gnutella Protocol Specification v0.4," http://rfc-gnutella.sourceforge.net/, 2004.

[5] 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*, 2001, pp. 149–160.

[6] 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)*, vol. LNCS 2233, Springer, pp. 44–55, 2001.

[7] 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*, vol. 181, no. 1, pp. 3–15, 1997.

[8] C. Schmidt and M. Parashar, "Enabling flexible queries with guarantees in P2P systems," *Internet Computing, IEEE*, vol. 8, no. 3, pp. 19–26, May-Jun 2004.

[9] 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*, 2004, pp. 353–366.

[10] 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 ComputingHPDC-12)*, Seattle, WA, June 2003.

[11] A. Iamnitchi, I. Foster, and D. Nurmi, "A Peer-to-Peer Approach to Resource Location in Grid Environments," *Proceedings of the 11th Symposium on High Performance Distributed Computing*, Aug 2002.

[12] W. Li, Z. Xu, F. Dong, and J. Zhang, "Grid resource discovery based on a routing-transferring model," *In Proceedings of the 3rd International Workshop on Grid Computing*, pp. 145–156, 2002.

[13] 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.

[14] M. Harchol-Balter, T. Leighton, and D. Lewin, "Resource Discovery in Distributed Networks," *In 17th Annual ACM Symposium on Principles of Distributed Computing*, pp. 229–237, 1999.

[15] S. Kutten and D. Peleg, "Asynchronous Resource Discovery in Peer to Peer Networks," *In 21st Symposium On Reliable Distributed Systems*, pp. 224–233, Oct 2002.

[16] I. Abraham and D. Dolev, "Asynchronous Resource Discovery," in *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, 2003, pp. 143–150.