Enabling Efficient Information Discovery in a Self-Structured Grid

Amos Brocco*[,1], Apostolos Malatras[1], Béat Hirsbrunner[1]

Pervasive and Artificial Intelligence Research Group, Department of Informatics, University of Fribourg, Boulevard de Pérolles 90, CH-1700 Fribourg, Switzerland

Abstract

One of the key success factors enabling the deployment of large scale grid systems is the existence of efficient resource discovery mechanisms. Accordingly, the main issues to be addressed by such a grid information system are those of scalability and minimal network overhead. In this respect, we propose a solution based on proactive information caching supported by a self-structured overlay topology. The proposed approach features a fully distributed ant-inspired self-organized overlay construction that maintains a bounded diameter overlay, and a selective flooding-based discovery algorithm that exploit local caches to reduce the number of visited nodes. To improve the caching scheme while retaining minimal bandwidth consumption, cache contents are periodically exchanged between neighboring nodes using an epidemic replication mechanism that is based on a gossiping algorithm, thus allowing nodes to have a more general view of the network and its resources. Extensive experimentation provides evidence that the average number of hops required to efficiently locate resources is limited and that our framework performs well with respect to hit rate and network overhead.

Key words: Grid Computing, Resource Discovery, Overlay Networks, Collaborative Ant Algorithms

1. Introduction

Sustained by the increasing availability of a large number of low cost high performance systems and high-speed network connections, distributed solutions are steadily emerging as a viable alternative to centralized approaches to solve complex computing tasks. Although centralized solutions are simpler to design and manage, they create a dependency of the whole system on a small number of machines, exposing a series of drawbacks such as increased load on the central nodes, and reduced robustness because of a single point of failure.

Distributed systems can be grouped in different categories according to their structure or purpose: peer-to-peer systems, ad-hoc networks, mesh networking, grid systems, etc. Despite some differences, the common goal driving the development of distributed systems is the ability to share local resources with the community, for example data in peer-to-peer systems or computing resources in grids. In particular, this paper focuses on resource discovery in computing grids. Grid resources are described by a resource profile listing the capabilities of the resource according to some metrics. Conversely, discovery queries list the needed capabilities of the requested resource. Provided that a matching mechanism to compare profiles to queries is available, the main issue to be addressed is the actual discovery of these profiles in a distributed environment.

Resource discovery in grid systems typically relies on a client-server model using centralized directories, where profiles are being published and incoming queries are being matched. This approach enables efficient and deterministic search, but requires large and powerful systems to store all the amount of information generated by the network. Although these issues do not hinder the inherent processing capabilities of a distributed computing system, they could negate some of the typical advantages offered by fully distributed models such as increased reliability and robustness. Additionally, if the information about resources changes rapidly, then the quality of resource discovery may be affected negatively, unless frequent updates are made to the central index.

Tackling the problem from a different perspective, other approaches realize the need to adopt solutions that explicitly take into consideration the nature of distributed systems so as to accommodate the grid resource discovery needs in a more coordinated manner, and take full advantage of the benefits of distribution. On the downside, because in this view all kind of centralized indexes are to be avoided, distributed mechanisms enabling efficient and effective discovery need to be developed. In this respect, peer-to-peer architectures are the most prominent paradigm that addresses grid resource discovery in a fully distributed manner. In particular, it is possible to recognize two main approaches: structured and unstructured.

Structured solutions propose the construction of topolo-

*Corresponding author

Email addresses: amos.brocco@unifr.ch (Amos Brocco), apostolos.malatras@unifr.ch (Apostolos Malatras), beat.hirsbrunner@unifr.ch (Béat Hirsbrunner)

gies with strict properties that allow deterministic resource discovery. On these systems, referred to as Distributed Hashtables (DHT) [1, 2], the overlay network is organized in such a way so that information can be easily located by means of keys associated to the underlying network node identifiers. While achieving satisfactory results in terms of resource discovery efficiency, there are limitations on the characterization of indexed resources. Specifically, grid resource discovery queries do not typically match to a unique key, as required by DHT, introducing therefore a degree of difficulty in locating complex combinations of hardware and software configurations [3].

To overcome the rigidness of structured solutions, an alternative approach involves the use of overlay networks without a fixed topology that can be classified as unstructured. Unstructured solutions do not have strict rules about the topology of the network nor about the location of resources. Resources can be located by means of flooding mechanisms, namely first by querying neighbors of the local node and then propagating these queries progressively throughout the network. Despite their advantages over structured systems, scalability of unstructured solutions may be limited by the exponential growth of flooding-related traffic. To address this problem, more efficient flooding methods have been subsequently developed, e.g. selective flooding [4], random walks [5], routing indices [6], semantic overlays [7, 23], etc.

In general, we consider ensuring fully distributed operation, limiting network overhead and minimizing response time, as fundamental requirements for grid resource discovery. In this regard, this paper presents a distributed grid information system supported by swarm intelligence for efficient resource discovery using flooding-like protocols. The proposed framework utilizes ant colony algorithms to build, optimize and maintain a self-structured peer-to-peer overlay network connecting grid nodes both using a minimal number of links, and ensuring that the diameter of this overlay network is bounded. We further augment this framework in terms of resource discovery efficiency by proactively querying the overlay network with the purpose of locating nodes with similar capabilities and storing this information in a local cache for every node, so as to minimize the amount of queries being propagated throughout the network to find matching nodes.

The remainder of this paper is structured as follows: Section 2 discusses related research in the field of resource discovery in unstructured overlay networks. Section 3 details the self-organized overlay construction algorithm, whereas Section 4 presents our proposed proactive resource discovery mechanism. The evaluation methodology for both algorithms is discussed in Section 5, while results and their analysis are presented in Section 6. Finally, Section 7 summarizes the work presented in the paper and provides some insight on future research directions.

2. Related Work

Improving the efficiency of resource discovery in unstructured peer-to-peer networks has been the subject of several research projects. The main goal of these projects is to reduce the overall traffic by minimizing the total number of nodes visited by a query. Although many distributed systems resource discovery mechanisms can be equally applied to peer-to-peer and grid systems as they share many principles, a distinction between these systems is in the definition of the notion of the resource. Whereas in peer-to-peer systems resources typically refer to files being shared amongst nodes, in grid systems it is computing resources that are being shared. In grid systems it is thus beneficial to collect as many query responses as possible so as to have a large selection of prospective candidate nodes to assign grid tasks to, whereas in peer-to-peer systems it is sufficient to have a small number of successful responses.

An important aspect of the proposed solution is the use of a self-organized optimized peer-to-peer overlay, in order to limit the traffic overhead resulting from the resource discovery process. This goal can be achieved by limiting the average path length in the overlay, as well as keeping a minimal number of links between nodes. A similar approach is followed by Umm [24], by distinguinshing between overlay maintainance and efficient information dissemination. In this respect, connections between nodes are optimized in order to reduce latency and to increase available bandwidth. Overlays with bounded diameter can be obtained in a fully distributed way by approximating a random or small-world topology, as shown by Newscast [25]. An example of a communication platform exploiting an optimized overlay is the Saxons project [26], where an overlay with both low latency and high bandwidth paths is constructed in order to provide efficient multicast delivery. A further example of self-organization is P-Grid [27], which exploits a dynamic DHT-like structure to index information and ensure fair replication of data across the nodes.

Efficiency of resource discovery in unstructured networks has been thoroughly analyzed in [8]. In general, we can identify two main categories of approaches aimed at optimizing the resource discovery process: solutions that do not consider the semantic of shared resources, and semantic-aware solutions.

The former category spans from expanding ring or random walks [5], to probabilistic flooding [9], teeming [10], or replication [11], etc. The underlying concept of these approaches is to reduce the number of query forwardings by limiting the spread of a query (for example, by setting small time-to-live values or by forwarding the query only to a limited random subset of the nodes) or by replicating information across the network. Different replication strategies may be adopted, such as path replication or uniform distribution. Although these methods effectively reduce the network load, we argue that a semantic-aware solution can be even more efficient.

Semantic-aware solutions exploit the information concerning both the resources and the queries in order to drive the request toward nodes that are more likely to fulfill it. Example solutions include [12, 13], which suggest using local indices to direct search queries toward nodes that are more likely to satisfy them. The forwarding policy is normally based on *satisfaction* indices that are evaluated based on past experiences, namely successful query responses. Upon this model, different propagation strategies can be implemented, as suggested in [14]. Following similar ideas, [15] describes a grid information service based on peer-to-peer technologies that uses routing indices to direct queries toward the closest known node that might fulfill the request. The same project also makes use of a super-peer topology, with nodes belonging to the same virtual organization connecting to one or more super-peers, thus further reducing the generated traffic.

An interesting solution is proposed by ANTARES [16], where a distributed swarm intelligence algorithm is used to cluster node references in a grid. ANTARES works on the principle of disseminating information about available resources across the network and uses clustering to reduce the traffic overhead. This solution exploits semantic information about resources to enhance proximity between nodes with similar profiles. When a query reaches a matching node, it is possible to find additional hits at a reduced cost.

Building on some of the concepts of the previously presented approaches, our framework follows a twofold approach by first optimizing the overlay network and then by exploiting local indices to improve resource discovery efficiency. In contrast to clustering solutions (such as ANTARES), local caches do not rely on vicinity between nodes in the overlay, and thus are more suited for self-organized overlays that cannot guarantee a stable topology.

3. Overlay Management

The overlay management provides connectivity between nodes upon an optimized overlay with minimal average path length, maintained by a self-organized collaborative algorithm named BLÅTANT-S. BLÅTANT-S improves our previous algorithm BLÅTANT-R [17] by simplifying its operation and reducing the overall network traffic, while retaining its quality behavior. A brief discussion of the differences between these two versions of the algorithm is provided at the end of this section. The algorithm depends on different species of ant-like software agents that move across the network and optimize its topology both by adding new logical links required to reduce the diameter, and also by removing existing links that do not contribute to the solution. The behavior of these mobile agents is inspired by swarm intelligence [18]. This section provides an overview of the BLÅTANT-S algorithm.

3.1. Peer Logic

The proposed overlay management algorithm is fully distributed across network peers. Each peer n_i contributes to the optimization of the network by rearranging local links according to two simple rules for connections and disconnections, which depend both on partial local view of the overlay and a user-defined optimization constraint parameter D.

Connection Rule. Consider two non-connected peers n_i and n_j in an overlay network G, and $d_G(n_i, n_j)$ the minimal routing distance from n_i to n_j in G. A new logical connection between n_i and n_j is created if:

$$d'_G(n_i, n_j) \geq 2D - 1 \quad (1)$$

Where $d'_G(x, y)$ is defined as $min(d_G(x, y), d_G(y, x))$.

The Connection Rule triggers the creation of new logical links, which ultimately reduce the diameter of the network to values $< 2D - 1$. Conversely, a Disconnection Rule is used to remove redundant links that are not necessary to bound the diameter.

Disconnection Rule. Consider two connected peers n_i and n_j in an overlay network G, $i \neq j$. Let $G' \leftarrow G \setminus \{n_i\}$ and N_i be the set of all nodes adjacent to n_i. Peer n_i is disconnected from $n_j \in N_i$ if:

$$\exists\, n_k \in N_i, k \neq j, |N_j| > |N_k| \;:\; d^*_{G'}(n_j, n_k)+1 \leq D(2)$$

Where $d^*_G(x, y)$ is defined as $max(d_G(x, y), d_G(y, x))$.

With a global knowledge of the network, even a distributed application of both rules leads to an optimized overlay with diameter d, $D \leq d < 2D - 1$. In order to discover other peers matching these rules, each peer n_i maintains a partial view of the network in a fixed-size table α_i, which retains neighborhood information. This information is continuously updated using the data coming from other nodes, and is used to evaluate the need for new links between two nodes, or the redundancy of existing connections. It should be noted that in fully distributed highly dynamic scenarios, diameter boundaries might only be approximated: according to our experiments, distances close to the $2D - 1$ boundary cannot be easily observed; nonetheless, the average path length will still converge to a value around $2D - 1$. A more detailed review of the aforementioned rules along with proofs of convergence is available in [17].

Each peer n_i maintains a set of identifiers of peers inside its neighborhood set N_i. Accordingly, two nodes n_i and n_j are considered as *connected* when both $n_i \in N_j$ and $n_j \in N_i$. In order to avoid hubs, the maximum size for the neighborhood set is limited, thus forcing the algorithm to optimize the network by re-arranging existing links instead of creating a large number of connections to all but a small number of peers. To support fault tolerance, each time N_i gets updated, all neighbors $n_k \in N_i$ get a notification from n_i, and update their respective α_k table.

3.2. Ant Agents

Most of the activities required for the management of the overlay are carried out by ant-like mobile agents. We distinguish between different classes of ants, or *ant species*, depending on the assigned task.

Discovery Ants are used to collect information about the network and to update the α table on each peer. Each agent wanders randomly across the network carrying a fixed-size circular buffer where identifiers of visited peers are stored. *Discovery Ants* have a limited lifespan, and are respawned by nodes at regular intervals according to a defined per-node birth probability.

Construction-Link Ants are sent by a peer n_j wanting to connect to the overlay. If the recipient has already reached the maximum number of allowed links, the ant is forwarded to the neighbor with the lowest degree. When some peer n_i accepts a connection request, the requesting peer n_j is added to the neighborhood N_i set and the ant is sent back to n_j, where n_i is conversely added to N_j.

Optimization-Link Ants are used to create links between nodes according to the Connection Rule. Similarly to construction ants, a peer n_j wanting to connect to a peer n_i sends an ant to the latter, but in contrast to what occurs with other species, n_i cannot forward the request to its neighbors, but can just accept or reject it.

Unlink Ants remove existing links between peers either because the Disconnection Rule applies, or because one of the peers wants to leave the network. An *Unlink Ant* travels to its target peer and removes all information about the source peer from the local α table and neighborhood set N_i.

Update Neighbors Ants are generated by a node n_i when its neighborhood set N_i changes. These ants travel to every neighbor $n_j \in N_i$ and update the information about n_i in the respective α_j tables.

Ping Ants are periodically exchanged between nodes to keep connections alive in low traffic situations.

3.3. Rules Evaluation.

Discovery Ants relay the collected information to each visited node. This information can be represented as a vector v of identifiers of nodes as visited by the ant. A node n_i receiving a vector v updates its α_i table, and then proceeds by determining if the *Disconnection Rule* applies. The information vector v is traversed and the distance d_v between each pair of identifiers n_j, n_z with $z \neq j \ \wedge \ n_j, n_z \in N_i$ is determined (computed as the absolute indices difference). If $d_v < D$, a disconnection procedure is started by sending an *Unlink Ant* to the neighbor (either n_j or n_z) with the greatest degree (size of the neighborhood set). Conversely, to evaluate the *Connection Rule*, the distances of each node n_z in the local cache table are processed, and a connection procedure is started with the node at the greatest distance $\geq 2D-1$. Accordingly, an *Optimization-Link Ant* is sent to the selected node. To avoid sending multiple ants to the same node, if a connection procedure has already been started, the same node cannot be selected again within a pre-defined timeframe.

3.4. Pheromone Trails

Communication between real ants is achieved using a stigmergic (i.e. indirect) mechanism, which involves leaving chemical *pheromone* trails in the environment. These chemical traces can be sensed by other individuals in the colony and their concentration indicates the desirability of a given path. With time, unless new chemical is left by an insect of the colony, the concentration of the trail completely evaporates. Evaporation provides the benefit of seamlessly resolving errors and overcoming bad system decisions, which are less likely to be reinforced with time. In BLÅTANT-S, we use pheromone trails to ensure fair coverage of the network by *Discovery Ants*, and by simulating evaporation, trails are also used to keep track of whether a neighbor is still alive.

We simulate trails by defining a numerical value for each connection. When ant agents move between peers, they leave pheromone trails on both the starting and the destination node. Pheromone trail τ is reinforced according to the formula $\tau \leftarrow 1$. Conversely, evaporation updates the concentration of the trail as $\tau \leftarrow \tau * \psi$ with $\psi < 1$. On each node n_i, and for each neighbor n_j, a pheromone trail $\gamma[n_j]$ is reinforced by ants traveling from n_i to n_j. Discovery ants leaving peer n_i will then less preferably choose n_j, and instead follow a path to a neighbor with a corresponding trail of lower concentration.

For each neighbor n_j in N_i, a $\beta[n_i]$ trail is also reinforced by ants traveling from n_j to n_i. By monitoring the concentration of such trails, nodes can detect neighbors' abrupt disconnections, i.e. crashes. When a trail completely evaporates, the corresponding neighbor is assumed to have left the network and subsequently a recovery procedure is started. Because the network traffic alone may not guarantee that pheromone trails are properly reinforced, a mechanism to avoid complete evaporation is required. For this purpose, *Ping Ants* may be periodically exchanged between nodes. In particular, n_i node will send a *Ping Ant* to its neighbor n_j as soon as pheromone concentration on trail $\gamma_i[n_j]$ falls under a predefined threshold. This will reinforce both $\gamma_i[n_j]$ and $\beta_j[n_i]$.

3.5. Recovery Procedure

When a node n_i leaves the network, its neighbors must rearrange their connections in order to avoid partitioning. If n_i leaves the network properly, it can initiate the recovery procedure by itself. Conversely, if n_i unexpectedly quits the network (for example, because it crashed), all of its neighbors will start the recovery procedure as soon as the disappearance of n_i is detected (i.e. complete β pheromone evaporation).

The recovery procedure consists in sending Construction-Link ants to other known neighbors of n_i in order to re-establish proper connectivity among

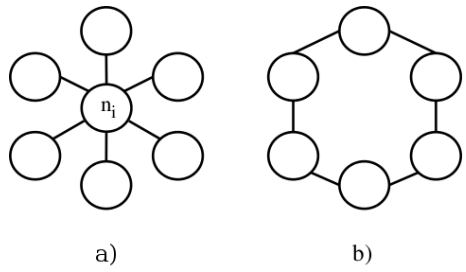

Figure 1: Recovery after node n_i leaves the network

them. In particular, ant agents try to connect all nodes in a ring formation, which requires the minimal number of links to ensure proper connectivity. Figure 1 depicts an example situation where node n_i leaves the network (a) and a recovery procedure is executed (b).

3.6. Comparison to BlåtAnt-R

In contrast to BlåtAnt-S, the former BlåtAnt-R version uses a more precise local evaluation of distances. In particular, instead of computing distance estimations from the relative positions of identifiers in v, a shortest-path algorithm [19] is used. Accordingly, more information needs to be collected by *Discovery Ants*, in order to enable the construction of a partial graph based on local data in the α table. Information vectors thus include not only the identifier of the node being visited, but also the identifiers of its neighbors. This evidently results in a bigger payload, which increases the overall traffic generated by the algorithm, as well as a greater computational complexity for the evaluation of distances.

4. Proactive Resource Discovery

The network overlay maintained by BlåtAnt-S enables optimized communication by providing bounded length paths between peers and by limiting the number of redundant paths between them. On one side, a small number of redundant links lowers the overhead created by multicast traffic. On the other side by actively bounding the diameter of the overlay, it is possible to fine-tune resource discovery query TTL (Time-To-Live, as hops in the overlay), thus limiting the spread of messages over the network. Meanwhile, obtaining satisfactory resource discovery hit rates still requires visiting a large number of nodes. In this respect, the aim of the proposed proactive resource discovery approach is to increase the hit rate by exploiting cached information in order to minimize network overhead.

4.1. Resource Discovery

Each node in the grid shares a set of its resources with other nodes, which can be referred to as the *resource profile* of the node. A resource profile can be viewed as a collection of tuples, expressed as a vector, referring to different resource aspects (i.e. CPU architecture, amount of memory, etc.) and their availability. We can distinguish between static and dynamic profiles: the former only considers the maximum theoretical availability of each resource, whereas the latter also considers the actual availability of each resource, which depends on the status of the node, the current active tasks (i.e applications running), and the scheduling policy. When users submit jobs to nodes that do not have the necessary resources to carry them out, resource discovery procedures are initiated. Resource discovery is the process of finding peers whose dynamic profiles match a given search query.

Resource discovery is performed using a limited and selective flooding algorithm. Limited flooding implies that nodes keep track of received queries, and avoid forwarding queries that have already been processed. Selective flooding means that, at each step, the query is forwarded only to a subset of all neighbors. In our approach, the subset is constructed by uniformly sampling the neighborhood set. We consider the query as *successful* when at least one node matching the query is found; conversely, each node found counts as a *hit*. Accordingly, the *hit rate* measures the percentage of successfully discovered resources out of all matching ones.

With our proactive caching approach we aim at exploiting the fact that similar nodes are more likely to match the same query. Unfortunately, there are many situations where an exact match might not be possible: it is thus necessary to determine resource profiles that, although different from the one required by the query, may still fulfill the task.

4.2. Peer Similarity

In order to determine if two peers share some similarities and are thus likely to match the same query we compare their resource profiles. In particular, given two resource profiles expressed as vectors, we use a *cosine similarity* measure.

Similarity Function Λ. Given two grid nodes n_i and n_j, their resource profile vectors p_i and p_j, a suitable scalar product operation, and a norm $\|.\|$, we consider a *similarity function* $\Lambda(p_i, p_j) \in [0, 1]$, such that

$$\Lambda(p_i, p_j) = \begin{cases} \frac{p_i \cdot p_j}{||p_i|| ||p_j||} & \text{if } \frac{p_i \cdot p_j}{||p_i|| ||p_j||} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The scalar product and the norm have to be defined such that the profiles are equivalent iff $\Lambda(p_i, p_j) = 1$, and *similar* iff this value is close to 1 according to a user-defined threshold (1% in our experiments).

4.3. Similar Peers Cache

Each node keeps a cache table of size c_{size} storing identifiers and timestamps of other nodes with a similar profile. In contrast to resource discovery, for caching purposes we

only consider static resource profiles. This cache is updated at regular intervals by starting *proactive* resource discovery queries to search for other nodes in the network having a similar profile. Results from proactive queries include both the identifier and the timestamp of the matching node. During proactive queries, each hit generates a reply message back to the originating node, in order to update its peer cache.

Similarly to routing indices [12], the collection of all peer caches can be viewed as a second-level overlay, where each node's neighborhood is composed of peers with similar resource profiles. Resource discovery is therefore enhanced because a pool of potential matching resources is immediately available.

4.4. Cache Merging

Maintaining up to date cache information through proactive resource discovery queries may lead to high network overhead. We thus introduce a cache merging mechanism that enables nodes to share their cache contents with peers having similar profiles. This avoids flooding the network with proactive queries, in favor of a pairwise exchange of a small number of node identifiers.

The process itself is inspired by the NEWSCAST [25] gossiping algorithm. At regular intervals, each node randomly chooses a peer from within its cache contents and initiates a merging procedure. The initiating peer requests the content of the remote cache, merges them with the local cache, and retains at most the $c_{size} - 1$ entries with the highest timestamp (i.e. the most recent information). Both the initiating node and the remote node will then replace their own caches with the resulting set. Finally, the initiating node will add the remote peer identifier, along with an updated timestamp to its cache. Conversely, the remote peer will add the initiating node's identifier and updated timestamp to its cache.

4.5. Enhanced Resource Discovery

The peer cache itself is exploited by non-proactive searches to enhance the hit rate: when a matching node is found instead of stopping the search, the query *jumps* to the node cache and continues for an additional number of steps. In this way, there is a high probability of reaching additional hits because of the way the cache has been constructed. Similarly to BLÅTANT-S ants, resource discovery queries also contribute in reinforcing β and γ pheromone trails as they propagate across the network.

5. Evaluation

A twofold evaluation of the proposed approach was conducted, focusing on both the performance of the overlay management algorithm as well as the efficiency of the enhanced proactive resource discovery mechanism. In particular, concerning the overlay we concentrated on the feasibility of maintaining a connected overlay with a bounded diameter even in dynamic topology scenarios. Regarding the assessment of the proposed resource discovery scheme, hit rate improvements as well as impact on bandwidth consumption were studied. Furthermore, a sensitivity analysis of different parameters of the proactive caching process was performed, in order to yield their influence on the aforementioned assessment metrics.

It should be noted that this work is an extension of our previous research documented in [21]; specifically, fine tuning optimizations were implemented in the proactive caching resource discovery protocol, enabling better overall performance, while additional scenarios were also considered.

5.1. Overlay Construction

To bootstrap the overlay, an initial random lattice consisting of 10 nodes was used; these nodes also form the pool of *well-known* connection points where any node wanting to join addresses its request to. At the beginning of the simulation a number of additional nodes is added, up to a total of 1281 nodes. The optimization parameter D for all scenarios is set to 5, thus the expected average path length is around $2D - 1 = 9$.

For evaluation purposes, tests were conducted in a discrete time simulator, with timing computed by means of *iterations*. It is nonetheless important to highlight the fact that execution of the algorithm remains fully distributed. At each iteration the whole population of ants may travel at most one hop in the overlay. Table 1 lists the parameters used by BLÅTANT-S during all simulations. Based on extensive experimentation on the overlay management algorithm [17], we argue that the presented choice of values represents an optimal configuration that provides satisfactory results. It should be nevertheless noted that different sets of values do not significantly affect the qualitative behavior of the presented results. Pheromone trails evaporation is simulated by updating their concentration at each iteration.

To simulate a dynamic network behavior, a new node joins the overlay with an average period of 50 iterations. Conversely, each 100 iterations a node leaves the network and one crashes (i.e. leaves the network abruptly). Consequently, the size of the network remains stable. In addition, nodes quitting the network overlay do not subsequently rejoin it.

To assess the impact of a different overlay management algorithm on the performance of the proposed caching mechanism, we also experimented with a topology constructed using NEWSCAST instead of BLÅTANT-S. For this experiment, each node maintained a NEWSCAST gossiping cache table of 20 entries, exchanged every 1000 iterations.

5.2. Resource Discovery Scenarios

Simulation of resource discovery is performed by randomly choosing both a starting node and a search profile. For evaluation purposes we only considered static

resources profiles. A set of 10 simulation runs of 75000 iterations each were evaluated: a set of 10 search queries is started every 25 iterations, beginning at iteration 500, resulting in 29800 queries per run. Different scenarios were simulated with varying values for the parameters of our resource discovery algorithm, as listed in Table 2. In particular, the considered parameters are noted as follows:

- **TTL**: resource discovery query time-to-live (hops);
- **FW**: selective forwarding sample size;
- **M-int**: cache merge interval;
- **P-int**: proactive queries interval;
- **C-TTL**: TTL while traveling within the cache;
- **C-FW**: FW within the cache;
- **P-TTL**: proactive queries TTL;
- **P-FW**: proactive queries FW;
- **C-Size**: cache size (maximum entries).

Each node is assigned a profile according to a uniform distribution, such that each profile is shared on average by 27 nodes. Accordingly, in all scenarios using the proactive caching, the cache size was set to 5 entries. The average path length in the overlay is 9, nonetheless the query TTL of all scenarios has been set to 5 in order to highlight the benefits of the cache mechanism while retaining a minimal traffic overhead.

5.3. Traffic Evaluation

In order to evaluate the traffic generated by our overlay management algorithm, we estimated the typical size of ant-like agents as follows:

- **Discovery**: 420 bytes *plus* 24 bytes/visited node;
- **Construction-link**: 420 bytes *plus* 24 bytes for the initiating node identifier;
- **Optimization-link**: 420 bytes;
- **Unlink**: 420 bytes;
- **Update Neighbors**: 420 bytes *plus* 24 bytes/neighbor.
- **Ping**: 420 bytes;

Accordingly, for the resource discovery task, we considered the following estimations:

- **resource discovery queries**: 1024 bytes;
- **resource discovery query replies**: 456 bytes;
- **cache merge**: 420 bytes *plus* 24 bytes/cache entry;
- **ping**: 352 bytes (ICMPv6).

These estimations include both the size of an IPv6 header (288 bytes), a UDP header (128 bytes), as well as a 4 bytes packet type identifier. For visited nodes, the assumed 24 bytes comprise 16 bytes for the IPv6 address, 4 bytes for the port number, and 4 bytes for the remote timestamp. The obtained traffic results are based on an average cost over the total number of queries, and include the overlay management, the proactive caching task (if applicable), and resource discovery. The bandwidth consumed by the overlay and caching does not depend on the resource discovery activity, and it should thus be considered as a fixed cost distributed among all queries.

Optimization parameter D	5
α table size	40
$max(\|N_i\|)\ \forall i$	8
Discovery Ant lifespan	50
Discovery Ant respawn interval	150
Discovery Ant birth probability	0.01
Discovery Ant vector length	20
Pheromone decay (for γ and β) ψ	0.991

Table 1: Overlay Construction Parameters

	TTL	FW	M-int	P-int	C-TTL	C-FW	P-TTL	P-FW	C-Size
A1	5	3	-	-	-	-	-	-	-
A2	5	4	-	-	-	-	-	-	-
A3	5	6	-	-	-	-	-	-	-
B1	5	3	2500	25000	3	3	4	3	5
B2	5	4	2500	25000	3	3	4	3	5
B3	5	6	2500	25000	3	3	4	3	5
C1	5	4	2500	25000	1	5	8	3	5
C2	5	4	2500	25000	2	5	8	3	5
C3	5	4	2500	25000	5	2	8	3	5
D1	5	4	10000	25000	3	3	8	3	5
D2	5	4	25000	25000	3	3	8	3	5
E1	5	4	2500	12500	3	3	8	3	5
E2	5	4	2500	37500	3	3	8	3	5
E3	5	4	2500	50000	3	3	8	3	5
F1	5	4	2500	25000	3	3	5	3	5
F2	5	4	2500	25000	3	3	6	3	5
G1	5	4	2500	25000	3	3	5	3	10
G2	5	4	2500	25000	3	3	5	3	20

Table 2: Evaluation Scenarios

6. Results

Building on the aforementioned evaluation scenarios and having detailed the considered parameters, we present and discuss here the corresponding results. The first set of results focuses on the performance of the overlay management algorithm, while the second one analyzes both the efficiency of the proposed resource discovery approach and the sensitivity of the caching algorithm to variation of parameters.

6.1. Overlay Management

We considered three different performance metrics to assess the proper operation and the efficiency of the overlay management algorithm: resulting average path length (AVPL), diameter and generated traffic. We compare the performance of the original BLÅTANT-R [17] algorithm with the improved BLÅTANT-S version by means of simulations. The latter algorithm has also been used for the resource discovery evaluation. Traffic results refer to the overlay management tasks only, and have been measured through simulations without resource discovery.

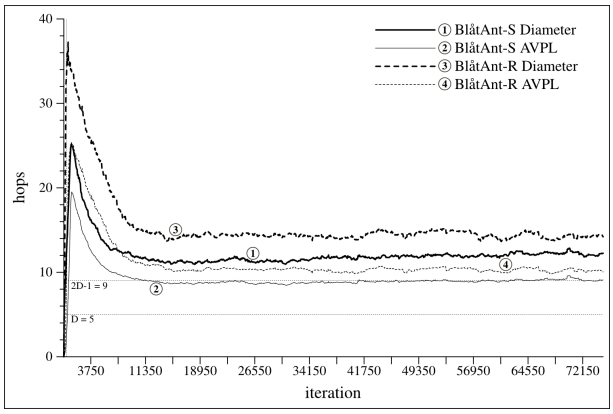

Figure 2: Average Path Length and Diameter

As shown in Figure 2, BlåtAnt-S obtains better results than BlåtAnt-R, both in diameter, and average path convergence with values close to 11 and $2D - 1 = 9$ respectively. The network bootstrap phase reflects on both the diameter and the average path length values, which increase up until around iteration 1000. As soon as all nodes are connected to the network, effects of the optimization become visible, and distances in the overlay quickly decrease.

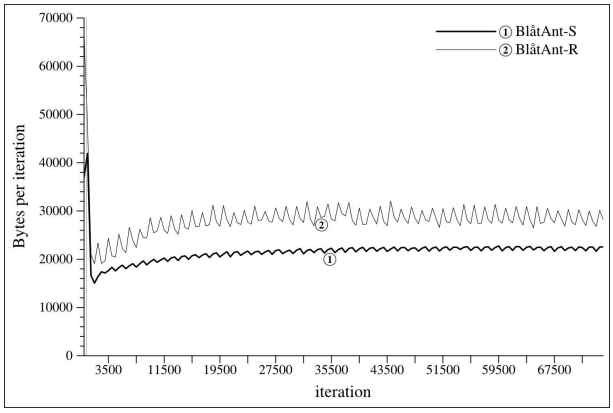

Figure 3: Overlay Management Traffic

As expected the improved algorithm consumes less bandwidth, with approximately 5 KB less traffic per iteration, as shown in Figure 3. It is worth noting that the generated traffic refers to the entire overlay, thus it averages to approximately 18 bytes per node per iteration for BlåtAnt-S. Although not shown in the figures, we also measured the total number of links in the resulting networks, namely 7400 for BlåtAnt-R, and 7000 for BlåtAnt-S. These results further confirm that overlays maintained by the improved algorithm are more optimized and contain less redundant links. While not illustrated, it has to be noted that during all simulation runs the network always remained fully connected, proving that the implemented recovery strategy is able to deal with the considered abrupt and proper disconnections rates.

6.2. Resource Discovery

Evaluation of the resource discovery efficiency has been conducted by means of the following assessment criteria: success rate (Figure 4), hit rate (Figure 5), cost per query (Figure 6, detailed in Figure 7), and cost per hit (Figure 8). Graphs concerning communication costs illustrate the effective resource discovery cost as well as the overlay and the proactive caching management costs. We consider a resource discovery query as being successful if at least one matching result is found, which also implies a hit rate greater than 0. Furthermore, each distinct matching node counts as a hit for the query. In the following we discuss the respective results for the diverse scenarios listed in Table 2.

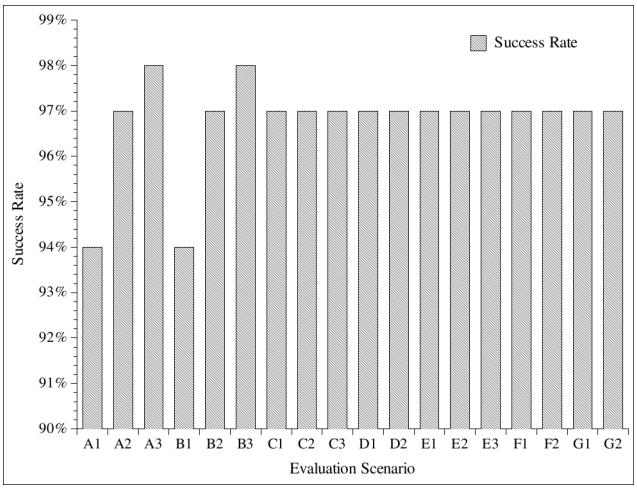

Figure 4: Search Success Rate

Without cache (A1,A2,A3). In order to have a baseline for comparison with our proposed proactive resource discovery mechanism, we experimented with different scenarios using only the limited and selective forwarding strategy. Furthermore, when a matching node is reached, the query is not further forwarded. As expected, the wider the spreading of the query, the more hits are found and the more traffic is generated. Although not shown, the cost of overlay management in the first scenario (A1) makes for a noticeably larger part of the overall traffic (an average of 24 KiB in A1, versus 14 KiB in A2/A3). The reason behind this behavior is the fact that a larger number of *Ping Ants* are exchanged between nodes because of the reduced application traffic (i.e. resource discovery). We also considered additional scenarios where forwarding was not stopped once a match was found, which did not however produce significant variations.

With cache (B1,B2,B3). The proactive caching strategy performs much better than the traditional approach. With only a slight increase in the average cost per query, it is possible to obtain a noteworthy improvement of the hit rate. In particular, as shown in Figure 5, the hit rate

is more than doubled in B1 in comparison to A1. As a consequence, the cost per hit in B1,B2,B3 is significantly reduced. Evidently the success rate remains unchanged in respect to the previous results, because cache information is exploited only once a match is found, and therefore only contributes to an increase of the hit rate. Scenario B1 shows the same behavior as A1, with respect to the overlay management traffic. It is also worth noting that B1 achieves almost the same hit rate as A2 while producing significantly less traffic. After having assessed the

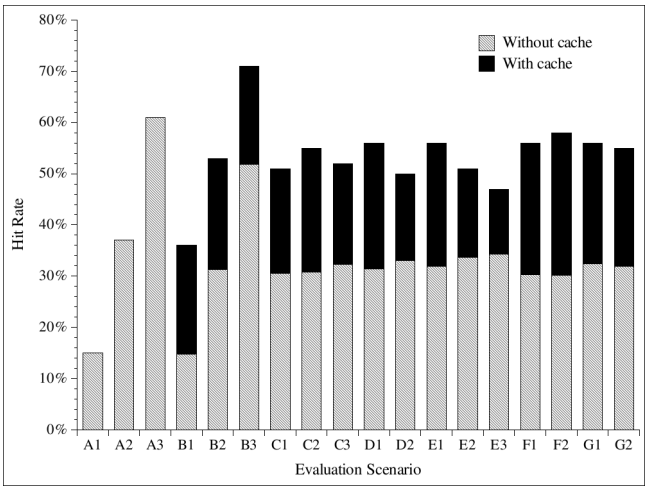

Figure 5: Search Hit Rate

improvements derived by our proactive caching approach, we perform a sensitivity analysis of the parameters affecting the caching behavior. In the following analysis we use scenario B2 as a baseline for comparison, being the most representative one.

Cache hops influence (C1,C2,C3). These scenarios are used to evaluate the impact of different cache navigation strategies (i.e. different C-TTL and C-FW). From the analysis of the results it is clear that no particular strategy significantly outperforms the others. Scenario C2 shows a slight improvement of the performance over B2 by reducing the cache TTL and increasing the cache FW. This result enables us to claim that similar nodes remain close to the node in the cache overlay, thus letting the query travel further in the cache does not provide any advantage.

Merge frequency influence (D1,D2). Cache merges promote spreading and sharing of information across the nodes as well as its replication. Additionally, merges also help removing old entries from the cache, thus avoiding referencing missing nodes that have already left the network. Decreasing the merge frequency lowers the amount of valuable information in the cache, which results in less hits being reported.

Proactive queries frequency influence (E1,E2,E3). Proactive queries are the primary mechanism used to fill the cache. As expected, the more frequent the proactive

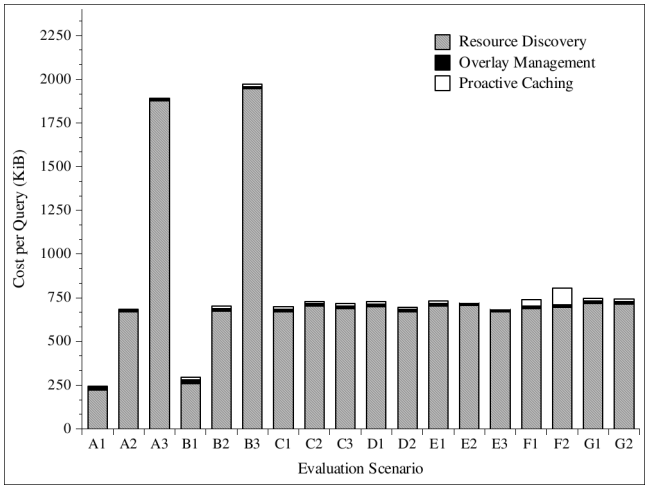

Figure 6: Cost per Query

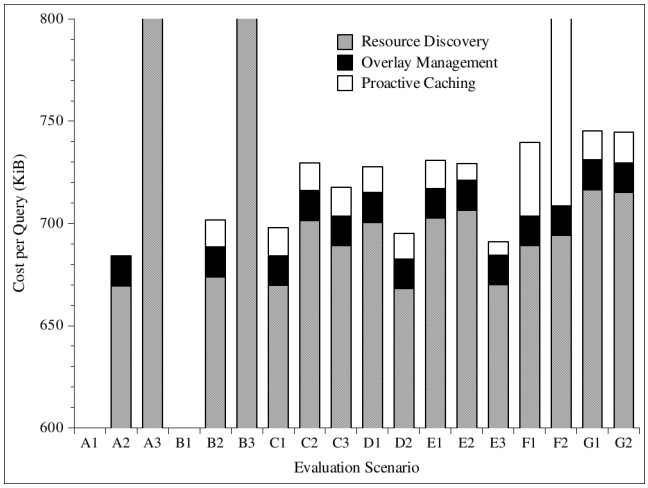

Figure 7: Cost per Query (detail)

queries are spawned on the network, the better the information in the cache is. In particular, scenario E1 realizes the best hit rate (56%) with the highest percentage of hits found in the cache (43%). Clearly, a counter-effect of higher frequencies is an increased cost affecting each query.

Proactive queries spreading (F1,F2). This result set concerns experiments with varying forwarding limits for proactive resource discovery queries. Obviously, if proactive queries travel deeper in the network, more hits are found at the expense of more traffic. Nonetheless, by comparing B2 and F2 (at a cost of 13 KiB and 98 KiB per query for proactive caching respectively), we can argue that the small benefits of an increased proactive query TTL do not compensate for the additional bandwidth consumption.

Cache size (G1,G2). The last result set concerns experiments with varying cache sizes. It is possible to notice that bigger caches improve the hit rate: from 53% in B2 with 5 cache entries, to 56% in G1 with 10, and 53% in G2 with 20. It is possible to notice a small regression from G1

to G2, because a large cache bears the risk of retaining a number of dangling node references.

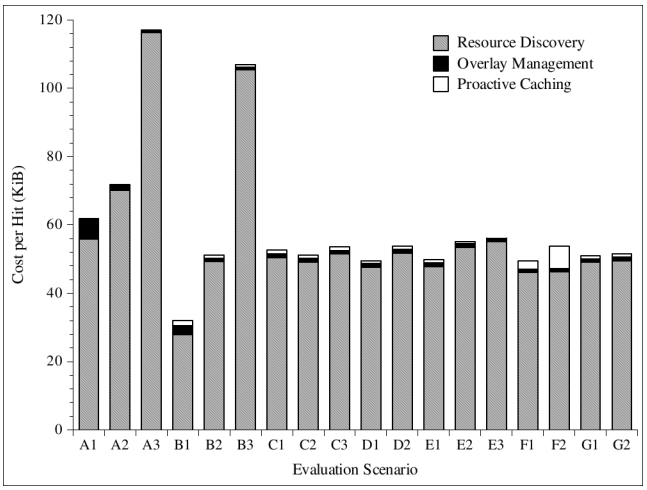

Figure 8: Cost per Hit

Overlay algorithm influence. The NEWSCAST scenario replicates the same resource discovery settings as the baseline B2. During the simulations, the observed average path length was between 5 and 6, whereas the diameter was approximately 10. These values are lower than the ones registered for BLÅTANT-S; nonetheless the different overlay management algorithm only slightly influenced the success rate, which dropped to 95%, and the hit rate, which remained at 53%. However, it is interesting to note that the contribution of the cache mechanism to the hit rate accounts for 55% of the overall results.

7. Conclusions

This paper presented an efficient resource discovery mechanism using proactive information caching supported by a gossiping protocol on a self-structured grid overlay. Regarding grid resource discovery, our driving motivation has been to achieve satisfactory hit rate results without imposing significant network overhead. The proposed scheme exploits a flooding-based protocol supported by proactive information caching. Caches are maintained by periodically executing proactive resource discovery in order to retrieve identifiers of other nodes with similar resource profiles. To further improve the performance of our system while limiting the network bandwidth consumption, we incorporated in our approach a gossip-based information spreading protocol that enables cache information sharing between nodes. We evaluated the aforementioned approach through extensive experimentation and assessed its merits compared to traditional flooding methods. In particular, we have been able to realize improvements in the hit rate with little impact on the generated traffic.

Concerning the grid overlay, we introduced a self-structured architecture maintained using a fully distributed ant-based algorithm, called BLÅTANT-S. The constructed topology ensures bounded average path length with minimal per-node degree and a minimal number of redundant links. Furthermore, we compared the behavior of the algorithm with its previous version, namely BLÅTANT-R, and established its improved performance both in respect of the quality of the generated overlay, and concerning network bandwidth requirements.

Compared to our original work presented in [21], we expanded our field of experimentation by including further evaluation scenarios, while in parallel fine-tuning the operation of the algorithm. It is noteworthy to mention that the proposed proactive caching mechanism is independent of the overlay management algorithm, and can thus be coupled with other overlay construction protocols. In this respect, an evaluation exploiting different network topologies would be of great interest. A first step towards the latter was the experiment with NEWSCAST. Future research will focus on adaptive strategies to control the proactive caching parameters, such as querying and merging intervals, according to monitored network conditions. Furthermore, a different application domain of the proposed framework that we plan to engage ourselves with is that of load balancing on grids. Although similar in principle to the problem tackled in this paper, different and more tailored solutions could be developed. This work is being developed in the context of the SMARTGRID project [22], which aims at exploiting collaborative and swarm intelligence algorithms in a grid management middleware.

8. Acknowledgments

This research has been carried out thanks to the financial support of the Swiss Hasler Foundation in the framework of the "ManCom Initiative", project Nr. 2122.

References

[1] Ion Stoica, Robert Morris, David Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 149–160, New York, USA, October 2001. ACM Press.

[2] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[3] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, May 2005.

[4] S. Arunkumar and R. S. Panwar. Efficient broadcast using selective flooding. In *INFOCOM*, pages 2060–2067, 1992.

[5] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: 16th Int. Conf. on Supercomputing*, pages 84–95. ACM, 2002.

[6] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. *22nd IEEE Int. Conf. on Distributed Computing Systems*, pages 23–32, 2002.

[7] Arturo Crespo and Hector G. Molina. Semantic overlay networks for p2p systems. Technical report, Stanford University, 2002.

[8] Saurabh Tewari and Leonard Kleinrock. Analysis of search and replication in unstructured peer-to-peer networks. In *SIGMETRICS '05: ACM SIGMETRICS Int. Conf. on Measurement and modeling of computer systems*, volume 33, pages 404–405, New York, NY, USA, June 2005. ACM Press.

[9] Vana Kalogeraki, Dimitrios Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *CIKM '02: 11th Int. Conf. on Information and knowledge management*, pages 300–307, New York, USA, 2002. ACM.

[10] Vassilios V. Dimakopoulos and Evaggelia Pitoura. On the performance of flooding-based resource discovery. *IEEE Trans. Parallel Distrib. Syst.*, 17(11):1242–1252, 2006.

[11] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. *SIGCOMM Comput. Commun. Rev.*, 32(4):177–190, 2002.

[12] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *22nd Int. Conf. on Distributed Computing Systems (ICDCS'02)*, pages 5–13. IEEE, 2002.

[13] Vicent Cholvi and Pascal Felber. Efficient search in unstructured peer-to-peer networks. In *European Transactions on Telecommunications: Special Issue on P2P Networking and P2P Services*, page 2004, 2004.

[14] Adriana Iamnitchi and Ian Foster. A peer-to-peer approach to resource location in grid environments. *Grid resource management: state of the art and future trends*, pages 413–429, 2004.

[15] Diego Puppin, Stefano Moncelli, Ranieri Baraglia, Nicola Tonellotto, and Fabrizio Silvestri. A grid information service based on peer-to-peer. In Jos C. Cunha and Pedro D. Medeiros, editors, *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 2005.

[16] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. Antares: an ant-inspired p2p information system for a self-structured grid. In *BIONETICS 2007 - 2nd Int. Conf. on Bio-Inspired Models of Network, Information, and Computing Systems*, Budapest, Hungary, December 2007.

[17] Amos Brocco, Fulvio Frapolli, and Béat Hirsbrunner. Bounded diameter overlay construction: A self organized approach. In *IEEE Swarm Intelligence Symposium*. IEEE, April 2009.

[18] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[19] Edsger W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[25] Márk Jelasity and Maarten van Steen. Large-scale newscast computing on the internet. Technical Report IR-503, Vrije Universiteit, Amsterdam, October 2002.

[21] Amos Brocco, Apostolos Malatras, and Béat Hirsbrunner. Proactive information caching for efficient resource discovery in a self-structured grid. In *Workshop on Bio-Inspired Algorithms for Distributed Systems*. ICAC 2009, ACM, June 2009.

[22] Ye Huang, Amos Brocco, Béat Hirsbrunner, Michèle Courant, and Pierre Kuonen. Smartgrid: A fully decentralized grid scheduling framework supported by swarm intelligence. In *7th Int. Conf. on Grid and Cooperative Computing*. GCC2008, October 2008.

[23] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186, New York, NY, USA, 2003. ACM.

[24] M. Ripeanu, A. Iamnitchi, I. Foster, and A. Rogers. In search of simplicity: A self-organizing group communication overlay. *SASO '07*, pages 371–374, July 2007.

[25] Márk Jelasity and Maarten van Steen. Large-scale newscast computing on the internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, October 2002.

[26] K. Shen. Structure management for scalable overlay service construction. In *NSDI'04*, pages 21–21. USENIX Association, 2004.

[27] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.