Proactive Information Caching for Efficient Resource Discovery in a Self-Structured Grid

Amos Brocco
Department of Informatics
University of Fribourg
Fribourg, Switzerland
amos.brocco@unifr.ch

Apostolos Malatras
Department of Informatics
University of Fribourg
Fribourg, Switzerland
apostolos.malatras@unifr.ch

Béat Hirsbrunner
Department of Informatics
University of Fribourg
Fribourg, Switzerland
beat.hirsbrunner@unifr.ch

ABSTRACT

The cornerstone of successful deployment of large scale grid systems depends on efficient resource discovery mechanisms. In this respect, this paper presents a grid information system supported by a self-structured overlay topology and proactive information caching. The proposed approach features an 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. The caches 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. We conducted extensive experimentation that 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.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms

Algorithms, Performance, Experimentation

Keywords

Grid Computing, Resource Discovery, Overlay Networks, Collaborative Ant Algorithms

1. INTRODUCTION

In recent years, there has been an established shift from centralized systems, networks and architectures towards decentralized approaches. Despite being simpler to design, centralized solutions suffer from inherent deficiencies, such as the existence of a single point of failure and therefore diminished reliability, increased load overhead on centrally located entities, and lack of flexibility and extensibility because of rigid system design. These drawbacks have also spurred developments in the field of distributed computing. Owing to the availability of low cost high performance computing systems, coupled with increased network bandwidth, distributed computing systems are now being considered as realistic alternatives to their traditional centralized counterparts. Nonetheless, many of these systems may not yet be considered as fully distributed, as their operation still depends on centralized services.

Distributed computing systems span from peer-to-peer systems, to ad hoc networks, mesh networking and also grid systems. The main concept behind the aforementioned systems is the collective sharing of resources, whether they refer to data in peer-to-peer systems or computing resources in grid systems. When resources are distributed among different grid nodes, a mechanism enabling efficient and effective discovery is required. In particular, this paper focuses on resource discovery in computing grids.

Initial attempts to solve this problem were towards the direction of porting the centralized computing paradigm in the distributed realm, namely exploiting periodically updated centralized indices that hold all resource location information, e.g. Napster, and can be accessed by any interested party. Nonetheless, such solutions require large and powerful systems to store all the amount of information generated by the network, while they also bear the aforementioned deficits of centralized systems; 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. More recent approaches realize the need to adopt solutions that explicitly take into consideration the nature of distributed systems so as to accommodate the resource discovery needs in a more coordinated manner, and take full advantage of the benefits of distribution.

In this respect, as far as fully decentralized resource discovery is concerned, two research streams have been proposed. Both are based on the principle of building an overlay network that assists in scaling down the complexity and size of the underlying peer-to-peer or grid system. A first approach suggests the construction of structured network topologies, i.e. topologies with fixed properties. 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. Unfortunately, not all resources can be easily tagged with a unique key, thus complex resources such as combinations of hardware and software configurations, as used in grids, may be difficult to index using a DHT [3].

A second approach, to overcome the strictness of structured solutions, involves the use of overlay networks without a fixed topology that can be classified as unstructured. Unstructured solutions do not have rigid rules about nodes joining or leaving the network nor about the location of resources. Nodes locate resources using flooding mechanisms, namely first by querying their neighbors and then propagating these queries progressively throughout the network. While initial approaches deploy network flooding to achieve resource discovery, the lack of scalability of such an approach and its inverse effect on increasing network overhead has led to the adoption of more efficient methods to resolve resource queries, e.g. selective flooding [4], random walks [5], routing indices [6], semantic overlays [7], etc. Generally speaking, a fundamental requirement and concurrently an assessment criterion regarding resource discovery for grid applications is to ensure limited network overhead and minimal response time.

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

Different research projects strive to improve the search efficiency in unstructured networks by reducing the number of nodes visited by a query. As we are only interested in smart forwarding techniques, we will not consider solutions that are unaware of the semantic of queries (i.e. expanding ring or random walks [5]). We therefore restrain our focus to routing indices, experience-based query forwarding, and clustering.

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.

Efficiency of resource discovery in unstructured networks has been thoroughly analyzed in [8]. By replicating information across the network, the chances of success are increased. Different replication strategies may be adopted, such as path replication or uniform distribution. Concerning the grid, an interesting solution is proposed by ANTARES [9], where a distributed swarm intelligence algorithm is used to cluster node references. This solution enhances proximity between nodes with similar profiles, thus reducing the cost of finding additional hits once a result is found.

Other solutions [10, 11] 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 [12]. Following similar ideas, [13] 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.

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.

3. OVERLAY MANAGEMENT

The network overlay is maintained by a self-organize collaborative algorithm named BLÅTANT-S, which improves our previous algorithm BLÅTANT-R [14] by reducing the overall network traffic while retaining its quality behavior. 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. This section provides an overview of the BLÅTANT-S algorithm.

3.1 Peer Logic and Data Structures

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:

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

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:

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

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; nonetheless, the average path length will still converge to a value around $2D-1$. A more detailed review of the aforementioned rules is available in [14].

Furthermore, each peer n_i maintains a set of identifiers of peers inside its neighborhood set N_i: 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, 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 are notified by n_i.

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 just accept or reject it.

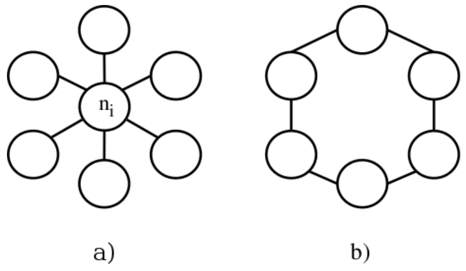

Figure 1: Recovery after node n_i leaves the network

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 Pheromone Trails

When ant agents move between peers, they leave pheromone trails on both the starting and the destination node. By simulating evaporation, these trails are used to keep track of whether a neighbor is still alive, as well as to ensure that the network is thoroughly explored, i.e. no parts of it are left unvisited. 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.

Pheromone trails are also used to detect abrupt node disconnections, i.e. crashes. For each neighbor n_j in N_i, a $\beta[n_i]$ trail is reinforced by ants traveling from n_j to n_i. When a trail completely evaporates, the corresponding neighbor is assumed to have left the network and subsequently a recovery procedure is started.

If the amount of traffic is not sufficient to keep pheromone trails from completely evaporating, ping ants are sent 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 $\beta_j[n_i]$.

3.4 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 them. In particular, ant agents try to connect all nodes with a minimal number of links by using a ring formation. Figure 1 depicts an example situation where node n_i leaves the network (a) and a recovery procedure is executed (b).

4. PROACTIVE RESOURCE DISCOVERY

The motivation behind the proposed approach for an optimized grid resource discovery mechanism, is the observation that the network overlay maintained by BLÅTANT-S provides bounded length communication paths between peers while retaining a low per-node degree. Being able to actively bound the diameter of the network allows to better fine-tune resource discovery query TTL (Time-To-Live, as hops in the overlay), thus limiting the overall traffic. Meanwhile, obtaining good hit rates would still require visiting a large number of nodes. Accordingly, the aim of the proposed proactive resource discovery approach is to increase the hit rate with minimal network overhead.

4.1 Peer Similarity

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. 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 profile matches a given search 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. In order to easily determine if two resource profiles, expressed as vectors, share some similarities, 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.

4.2 Similar Peers Cache

Each node keeps a cache table of size c_{size} storing identifiers and timestamps of other nodes with a similar profile. 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.

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.

4.3 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[15] gossiping algorithm. At regular intervals, each node chooses a peer at random 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 identifier and updated timestamp to its cache.

4.4 Resource Discovery

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*. During proactive queries, each hit generates a reply message back to the originating node, in order to update the peer cache.

The peer cache itself is exploited by non-proactive searches: 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

We conducted extensive evaluation of both the overlay management algorithm, and its combination with the resource discovery protocol. In particular, two main aspects of the system are considered: the ability of the overlay management algorithm to keep a connected overlay with a bounded diameter in a dynamic scenario, and the efficiency of resource discovery. Additionally, the impact on bandwidth consumption of both the overlay management tasks and the proactive caching in respect to the overall resource discovery is also studied.

5.1 Overlay Construction

The overlay is constructed starting from a random lattice consisting of 10 nodes, which constitute the pool of *well-known* connection points where any node wanting to join

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 ψ	0.991
Ping Ant threshold	0.25

Table 1: Overlay Construction Parameters

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 $2D - 1 = 9$.

Simulation timing is computed by means of *iterations*: at each iteration the whole population of ants may travel one hop in the overlay. Table 1 lists the parameters used by BLÅTANT-S during all simulations. Due to space limitations, the choice of these values is not discussed here; nevertheless, note that different sets of values do not significantly affect the qualitative behavior of the presented results. Pheromone evaporation is simulated by updating their concentration at each iteration.

To simulate a dynamic network behavior, a new node joins the network 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.

5.2 Resource Discovery Scenarios

Simulation of resource discovery is performed by randomly choosing both a starting node, and a random search profile. Several 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.

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 query TTL of all scenarios has been set to 9 because of the expected average path length as previously discussed.

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**: 444 bytes;

	TTL	FW	M-int	P-int	C-TTL	C-FW	P-TTL	P-FW
A1	9	3	-	-	-	-	-	-
A2	9	4	-	-	-	-	-	-
A3	9	5	-	-	-	-	-	-
A4	9	*all*	-	-	-	-	-	-
B1	9	3	2500	25000	3	3	8	3
B2	9	4	2500	25000	3	3	8	3
B3	9	*all*	2500	25000	3	3	8	3
C1	9	4	2500	25000	1	5	8	3
C2	9	4	2500	25000	2	5	8	3
C3	9	4	2500	25000	5	2	8	3
D1	9	4	10000	25000	3	3	8	3
D2	9	4	25000	25000	3	3	8	3
E1	9	4	2500	12500	3	3	8	3
E2	9	4	2500	37500	3	3	8	3
E3	9	4	2500	50000	3	3	8	3
F1	9	4	2500	25000	3	3	6	4
F2	9	4	2500	25000	3	3	9	4

Table 2: Evaluation Scenarios

- **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.

These estimations include both the size of an IPv6 header (288 bytes), and also a UDP header (128 bytes). 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.

6. RESULTS

Having detailed the parameters of the considered evaluation scenarios, we present and discuss here the corresponding results. The first part focuses on the performance of the overlay management algorithm, while the second part analyzes the efficiency of the proposed resource discovery approach.

6.1 Overlay Management

Three performance metrics have been considered to evaluate the overlay management algorithm: resulting average path length (AVPL), diameter and generated traffic. We compare the original BLÅTANT-R [14] algorithm with the improved BLÅTANT-S version, which 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.

As shown in Figure 2, BLÅTANT-S obtains better results than BLÅTANT-R, both in diameter, and average path con-

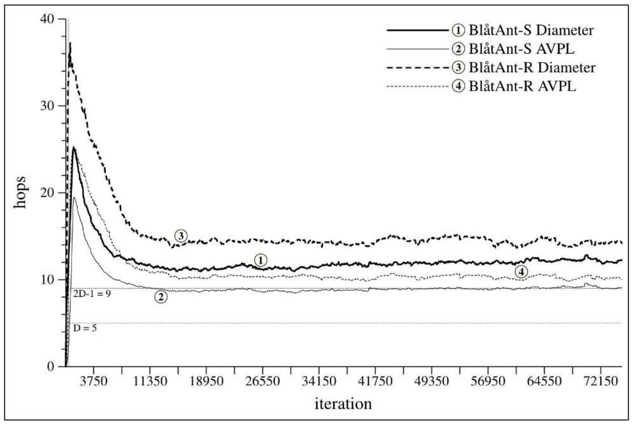

Figure 2: Average Path Length and Diameter

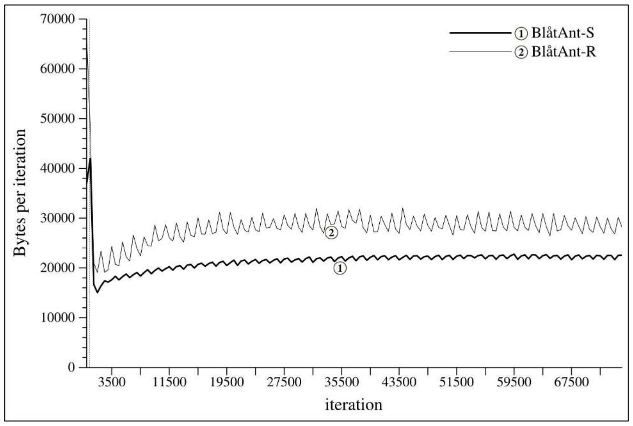

Figure 3: Overlay Management Traffic

vergence 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.

Additionally, as shown in Figure 3, the improved algorithm also consumes less bandwidth, with approximately 5 KB less traffic per iteration. 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.

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), and cost per hit (Figure 7).

Graphs regarding communication costs illustrate the effective resource discovery cost as well as the overlay and the proactive caching management costs. We consider a resource

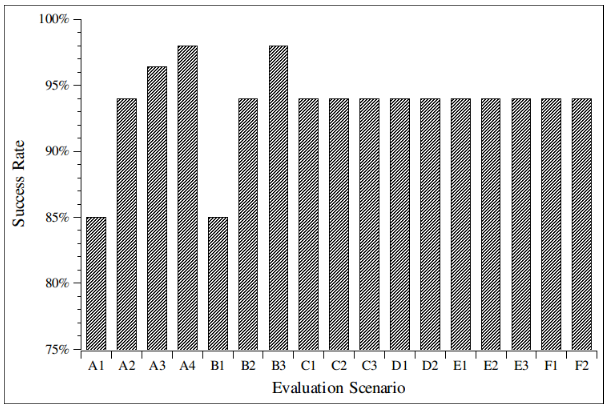

Figure 4: Query Success Rate

discovery query as being successful if at least one matching result is found; conversely, we count each distinct match as a hit.

Without cache (A1,A2,A3,A4).

Before assessing the performance of the proactive information caching strategy, we experimented with different scenarios using only the limited and selective forwarding strategy. If a query encounters a matching node it is not further forwarded. As expected, the wider the spreading of the query, the more hits are found and the more traffic is generated. In the first scenario, the cost of overlay management also makes for a noticeably larger part of the overall traffic: 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).

Although not depicted in the figures, we also evaluated 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. A slight increased average cost per query, resulted in a noteworthy increase in the hit rate, which almost doubled in B1 and B2 in comparison to A1, respectively A2. As a consequence, the cost per hit 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 to increase the hit rate. Scenario B1 shows the same behavior as A1, with respect to the overlay management traffic. It is worth noting that B2 achieves the same hit rate as A3 while producing significantly less traffic.

After having assessed the 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

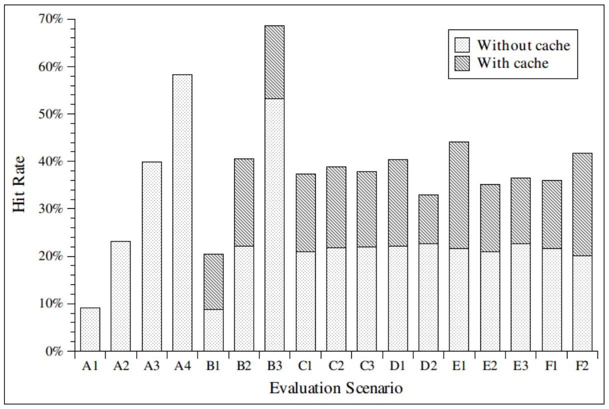

Figure 5: Search Hit Rate

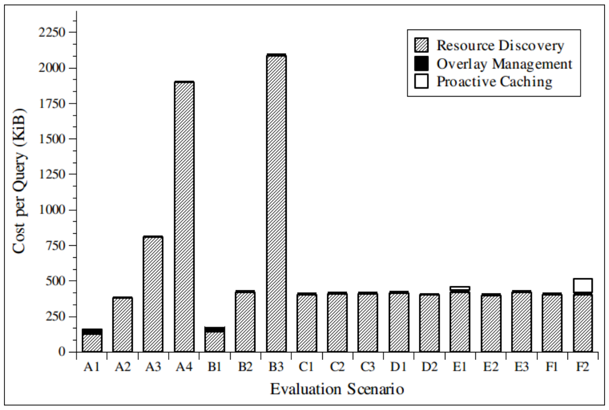

Figure 6: Cost per Query

no particular strategy significantly outperforms the others. Nonetheless, a balanced strategy, as in scenario B2, is confirmed as being optimal, achieving the highest hit rate.

Merge frequency influence (D1,D2).

Cache merges promote spreading and sharing information across the nodes as well as replication. Additionally, merges also help removing old entries from the cache, thus avoiding dangling references to 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).

As expected, the more frequent the proactive queries are spawned on the network, the better the information in the cache is. In particular, scenario E1 realizes the best hit rate (44%) with the highest percentage of hits found in the cache (51%). Clearly, a counter-effect of higher frequencies is an increased cost per query.

Proactive queries spreading (F1,F2).

The last 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, we can argue that the small

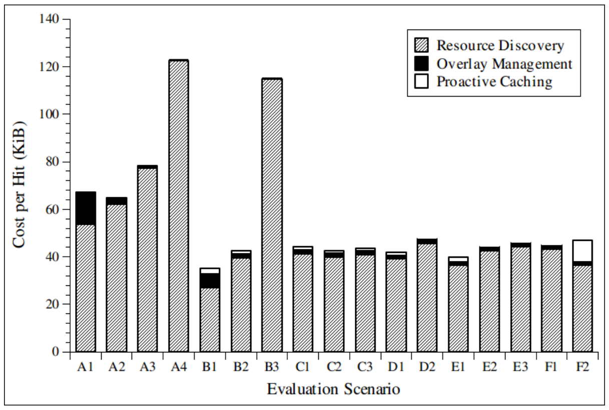

Figure 7: Cost per Hit

benefits of an increased proactive query TTL do not justify the additional bandwidth consumption.

7. CONCLUSIONS

In this paper we presented an efficient resource discovery scheme using proactive information caching on a self-structured grid overlay. The overlay is maintained using a fully distributed ant-based algorithm, called BLÅTANT-S, which ensures bounded average path length with minimal per-node degree. 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.

In regard to grid resource discovery, we recognized the need to achieve satisfactory hit rate results without imposing significant network overhead. We thus proposed an efficient flooding-based mechanism supported by proactive information caching combined with a gossip-based spreading protocol. 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.

Although the impact of overlay management and proactive caching on the overall traffic cost has shown to be negligible in the presented scenario, an evaluation exploiting different network scenarios would be of great interest. As such, future research will also 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 [16], which aims at exploiting collaborative 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.

9. REFERENCES

[1] I. Stoica, R. Morris, D. Karger, F. M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 149–160, New York, USA, October 2001. ACM Press.

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

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

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

[8] S. Tewari and L. Kleinrock. Analysis of search and replication in unstructured peer-to-peer networks. In *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, volume 33, pages 404–405, New York, USA, June 2005.

[9] A. Forestiero, C. Mastroianni, and G. Spezzano. Antares: an ant-inspired p2p information system for a self-structured grid. In *BIONETICS 2007 - 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, Hungary, December 2007.

[10] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 5–13, Washington, DC, USA, 2002. IEEE Computer Society.

[11] V. Cholvi and P. 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.

[12] A. Iamnitchi and I. 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.

[13] D. Puppin, S. Moncelli, R. Baraglia, N. Tonellotto, and F. Silvestri. A grid information service based on peer-to-peer. In Jose C. Cunha and Pedro D. Medeiros, editors, *Euro-Par*, volume 3648 of *LNCS*, pages 454–464. Springer, 2005.

[14] A. Brocco, F. Frapolli, and B. Hirsbrunner. Bounded diameter overlay construction: A self organized approach. In *IEEE Swarm Intelligence Symposium*. SIS 2009, IEEE, April 2009.

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

[16] Y. Huang, A. Brocco, B. Hirsbrunner, M. Courant, and P. Kuonen. Smartgrid: A fully decentralized grid scheduling framework supported by swarm intelligence. In *7th Int. Conf. on Grid and Cooperative Computing*. GCC2008, October 2008.