

A Scalable Architecture for Discovery and Planning in P2P Service Networks

Agostino Forestiero, Carlo Mastroianni
{forestiero,mastroianni}@icar.cnr.it
ICAR-CNR, Via P. Bucci 41C, 87036 Rende (CS), Italy

Harris Papadakis, Paraskevi Fragopoulou
{adanar,fragopou}@ics.forth.gr
Foundation for Research and Technology-Hellas (FORTH)
Institute of Computer Science N. Plastira 100
Vassilika Vouton, GR-700 13 Heraklion, Crete, Greece

Alberto Troisi, Eugenio Zimeo
{altroisi,zimeo}@unisannio.it
Department of Engineering, University of Sannio, Benevento, Italy



CoreGRID Technical Report
Number TR-0152
June 17, 2008

Institute on Knowledge and Data Management
Institute on Architectural Issues: Scalability, Dependability,
Adaptability

CoreGRID - Network of Excellence
URL: <http://www.coregrid.net>

A Scalable Architecture for Discovery and Planning in P2P Service Networks

Agostino Forestiero, Carlo Mastroianni
{forestiero,mastroianni}@icar.cnr.it
ICAR-CNR, Via P. Bucci 41C, 87036 Rende (CS), Italy

Harris Papadakis, Paraskevi Fragopoulou
{adanar,fragopou}@ics.forth.gr
Foundation for Research and Technology-Hellas (FORTH)
Institute of Computer Science N. Plastira 100
Vassilika Vouton, GR-700 13 Heraklion, Crete, Greece

Alberto Troisi, Eugenio Zimeo
{altroisi,zimeo}@unisannio.it
Department of Engineering, University of Sannio, Benevento, Italy

CoreGRID TR-0152

June 17, 2008

Abstract

The desirable global scalability of Grid systems has steered the research towards the employment of the peer-to-peer (P2P) paradigm for the development of new resource discovery systems. As Grid systems mature, the requirements for such a mechanism have grown from simply locating the desired service to compose more than one service to achieve a goal. In Semantic Grid, resource discovery systems should also be able to automatically construct any desired service if it is not already present in the system, by using other, already existing services. In this report, we present a novel system for the automatic discovery and composition of services, based on the P2P paradigm, having in mind (but not limited to) a Grid environment for the application. The report improves composition and discovery by exploiting a novel network partitioning scheme for the decoupling of services that belong to different domains and an ant-inspired algorithm that places co-used services in neighbouring peers.

1 Introduction

Future applications and services in Pervasive and Grid environments will need to support more and more distributed and collaborative processes. This requires that systems have the ability to cope with highly dynamic environments in which resources change continuously and in unpredictable ways, and might even be not existent when designing the system, thus calling for runtime discovery and composition.

While expectations on the quality of these systems are increasing dramatically, current methods, techniques, and technologies are not sufficient to deal with adaptive software in such dynamic environments. Large-scale systems will have to exhibit autonomic adaptation capabilities to adhere to emerging business, engineering and scientific processes.

Service-oriented architecture (SOA) represents a promising model for such dynamic environments. Services, in fact, are becoming important building blocks of many distributed applications where loosely connection among

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

components represents a key aspect to better implement functional distribution (i.e. contexts in which distribution is fundamental for implementing applications) and scalability (i.e. the ability to easily extend functional properties of a system). SOA can be usefully exploited in Grid computing to dynamically acquire available computational, communication or storage resources characterized by desired QoS offerings, to compose high-level functions for solving complex problems in a distributed environment or to enable virtual organizations to support collaborative research in e-Science.

Despite of its application in many domains, SOA-based technology still requires a lot of research to adopt it not only for increasing interoperability, but also to manage knowledge of pervasive resources and processes. In particular, the supervised approach for service discovery, composition, and execution (as with current service technology) can be a strong limitation since it represents a bottleneck from a performance point of view and imposes a centralized knowledge to discover services and build a composition topology in order to address a specific goal.

The report presents an approach based on P2P to overcome currently adopted connection and coordination models, which enables fully distributed and cooperative techniques for discovery, composition and enactment of services, optimized through semantic overlays. In particular, the report shows a technique that reduces the time for automatic service composition with respect to the centralized approach. The technique is mapped on P2P networks by exploiting two mechanisms for improving performance and scalability: (1) network partitioning to reduce message flooding and (2) an ant-inspired algorithm that allows for an efficient reorganization of service descriptors.

To restrict the search space, a partition mechanism is exploited. Different partitions correspond to different application domains, therefore the composition of a workflow for a specific application domain only needs the services placed in the corresponding partition. The ant algorithm exploits the operations of simple mobile agents that reorganize descriptors of services that are often co-used in composite services. This way, a service discovery procedure can discover the basic components of a composite service in a shorter time and with lower network traffic.

These two techniques, respectively proposed by FORTH [9] and CNR-ICAR [4] are integrated with a previous work implemented at the University of Sannio on P2P service composition [10] to improve its performance and scalability.

The rest of the report is organized as follows. Section 2 analyzes the state of the art in service discovery and composition by discussing related work. Section 3 presents a distributed planning technique to compose services in a workflow by exploiting a P2P model and discusses some possible mapping on P2P networks. Section 4 proposes a first improvement at network level by transforming an unstructured P2P network in a semi-structured one based on partitions that capture domains of knowledge in the service network. Section 5 shows an ant-inspired algorithm to aggregate descriptors of services related to the solution of the same problem or the most used services for solving a specific problem. Finally, Section 6 concludes the report and highlights future work among the CoreGrid partners to ultimate the integration in simulated and real environments.

2 Related Work

Service discovery is typically performed through centralized middleware components, such as registries, discovery engines and brokers. Even if this approach supports the necessary decoupling between producer and consumer objects or services, it will inevitably become a serious bottleneck when the number of services and organizations using services will grow. At high-level, in fact, the problem is particularly important since service discovery requires using sophisticated matchmaking algorithms to identify semantic similarities between consumers template and providers target descriptions. These algorithms consume many more computational resources, compared with other similar problems in the Internet such as DNS queries, and require distributed architectures for service registries and matchmakers able to scale to the growing number of service providers and consumers.

Distributed registries and registry federations have been proposed as a first approach to avoid bottlenecks in a service network. Some existing standards, in fact, adopt a federation of registries to satisfy some non-functional aspects of service discovery such as scalability and fault tolerance [11, 12]. METEOR-S [13] and PYRAMID-S [14], on the other hand, proposes a scalable P2P infrastructure to federate UDDI registries used to publish and discover services; it uses an ontology-based approach to organize registries according to a semantic classification based on the kind of domains they serve. Even though these solutions well address scalability and fault-tolerance through federation of registries, they use a structured topology of registries built on the basis of a domain-specific ontology. This enforces significant constraints on publication policies, impeding a fully exploitation of the P2P model.

In [15], the authors propose a distributed federation of registries coordinated by a publish/subscribe (P/S) infras-

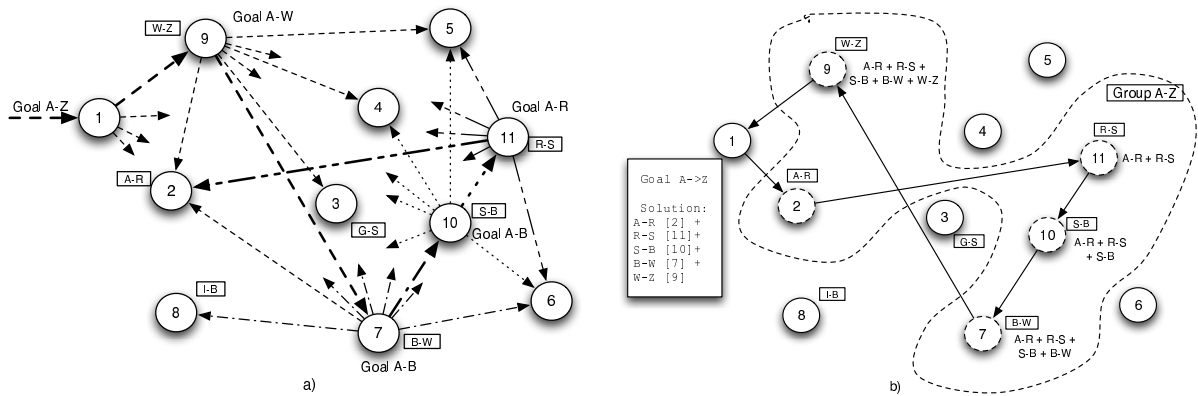


Figure 1: Cooperative composition of services to satisfy a goal

structure that is able to dispatch to the interested clients service availability as soon as a service is published in the network, by adopting a bus-oriented infrastructure to decouple registries. The proposed architecture is flexible but scalability is constrained by the number of organizations connected to the bus. Anyway, the infrastructure is mainly adopted to discover and not to compose existing services to solve a more complex problem than the ones directly solved with atomic published services.

Here, we propose a P2P model to generalize the adoption of P2P and cooperative approaches for service discovery, composition and enactment. This goes beyond existing P2P technologies such as Gnutella and JXTA, which implement a primitive form of service composition through the dynamic construction of paths through the network to address queries.

3 Self-organizing P2P Service Network

A service network should be able to self-organize in a dynamic and adaptive way in order to follow environmental changes and to structure the knowledge for continually optimizing service discovery and composition. In such a system of services, nodes should be able to communicate to find each other through discovery mechanisms that ensure high efficiency and scalability, with the aim of reducing response times. To this end, each node should be able to interpret an incoming goal and to give a partial or total contribution to the solution, even individuating other nodes in the network able to contribute with a piece of knowledge.

Figure 1 shows an example of cooperative composition in which node 1 injects in the network a goal (simply described in twith the implication $A \rightarrow Z$, which means that A is a pre-condition whereas Z is the post-condition to achieve) with the aim of discovering and composing the services whose execution changes the state of the system from A to Z .

3.1 Discovery and composition

We define a workflow composition with the deterministic transitional model that can be characterized by a finite set of states, a finite set of possible actions, an initial state and a goal state. Two states are connected if an operation exists that can be performed to transit from the first state into the second. In this model, the planning problem is equivalent to a state space search influenced by the strategy and the direction adopted. In the service domain, a state represents the set of necessary conditions for a service execution or the set of the effects produced by a service execution. An action is a possible execution and it can be defined with a pre-condition and a post-condition. So, the search of one or more solutions to a query has to find ordered collections of services that solve it.

Each node in the network contributes to discover the peers that can originate useful compositions. To this end, the nodes have the same computational capabilities and are interested in the same way by the query. They collaborate together in order to find a solution without any centralized coordination. According to the P2P model, peers become a crucial part of the architecture, since with this model the network lacks of structural components for discovery and composition.

Each peer is responsible of receiving requests (goals) from other nodes, and fulfilling them (i) by relying on service operations or lower level features available on each peer or (ii) by forwarding the request to other known peers (see Fig. 1 a). In many cases a peer can be able to fulfill a request by composing some of its operations with operations made available by other peers (see peers 1, 9, 7, 11). In such a case, a peer is also responsible of composing these operations, to fulfill either partially or totally the request received.

The algorithm adopted by each peer is described below:

```
search(Goal g){
  if peer has a solution for goal g
    send the solution to the goal's source;

  else if peer has a partial solution
    gapSolution = [distributed] search(partialGoal);
    solution = merge gapSolution with my partialSolution;
    send the solution to the goal's source;

  else
    return;
}
```

where the local search is performed by a semantic matching layer.

When a goal is resolved, the submitter peer receives either the composition of services (through the identifiers of services - see Fig. 1 b) or simply the identifier of the first service/peer to contact in order to start a distributed execution.

3.2 Network topology and overlays

When a composition is identified (see Fig. 1 b), the network implicitly aggregates the participating peers to form a new group that will simplify successive discovery and composition operations. This process is executed continually in the network, giving rise to several virtual layers of peers able to solve different problems at different abstraction layers. Therefore we can imagine having two distinct dimensions for the specialization of services: (1) a first dimension that organizes services according to the domain in which they are used, and (2) a second dimension that organizes services in groups to simplify new and more complex compositions.

Techniques based on network information limit the traffic in the network since queries are routed only to the peers that host the desired resource. In Distributed Hash Tables (DHT) for example, each peer or resource is mapped to a unique identifier in a known identifier space. The combination of unique identifiers and a known space from which these identifiers are drawn, allows routing to be achieved efficiently. The payoff for this efficiency however, is that such architectures require a highly structured network and do not well support ad hoc configurations. The strong constraints imposed to the publication of service advertisements severely limit the possibilities for cooperation among peer nodes. The lack of a single point of failure in P2P unstructured networks ensures a better fault tolerance of the overall system whereas the availability of a potentially non-limited storage capacity for indexing increases significantly network scalability. Unfortunately, unstructured networks typically adopt flooding to broadcast discovery messages to all the reachable peers connected to the network.

3.3 Experimental results

The network of services was simulated with the PeerSim P2P simulator [<http://peersim.sourceforge.net/>] (produced under the EU project BISON [<http://www.cs.unibo.it/bison/>]). To evaluate the performance with an increasing number of services in the network, the following scenarios were considered: centralized (i.e., all services published on a single peer), distributed (i.e., one service published for each peer) with a static network (reconfiguration disabled) and with a self-organizing network (reconfiguration enabled).

In the distributed scenario, the network was simulated by defining: an initial topology in which peers are grouped together in a generic group, the network view for each peer (i.e., the set of its neighbors peers/groups), the time to evaluate a matching between a service description and a query description (300 ms) and the messages transmission

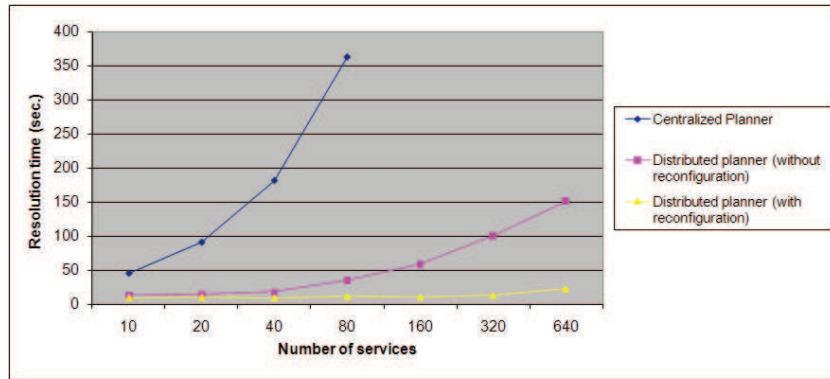


Figure 2: Planning speedup

delay (50 ms). The test cases are related to a goal satisfied by the composition of 10 specific services over a space of a variable number of services, that were randomly assigned to peers and affected the network dimension. Figure 2 shows that the distributed approach scales better than the centralized one. However, the speed-up over a static network is limited by its topology when a super-peer model is adopted. In general, it is difficult to statically define a network topology to enable the best search for each kind of goal. Better results can be achieved with a self-organizing network, in which peers with similar services can be grouped together on the basis of submitted goals, so forming during time specific domain areas.

The system has been also tested in a more realistic environment, defining a simple P2P network based on the JXTA middleware and integrating the planner module on each JXTA peer. The group and super-peer concepts have been preserved by specializing the concepts of JXTA Rendez-vous Peer and JXTA Group.

4 Network partitioning

One way to reduce the cost of flooding is to partition the overlay network into a small number of distinct subnetworks and to restrict the search for individual request to one network partition, thus reducing the overwhelming volume of traffic produced by flooding without affecting at all the accuracy of the search method (network coverage). The *Partitions* scheme, proposed in this section, enriches unstructured P2P systems with appropriate data location information in order to enable more scalable resource discovery, while not affecting at all the self-healing properties and the inherent robustness of these systems. The aim of the partitioning scheme is to improve the scalability of flooding by reducing the number of peers that need to be contacted on each request, without decreasing the probability of query success (accuracy of the search method) to the slightest extent.

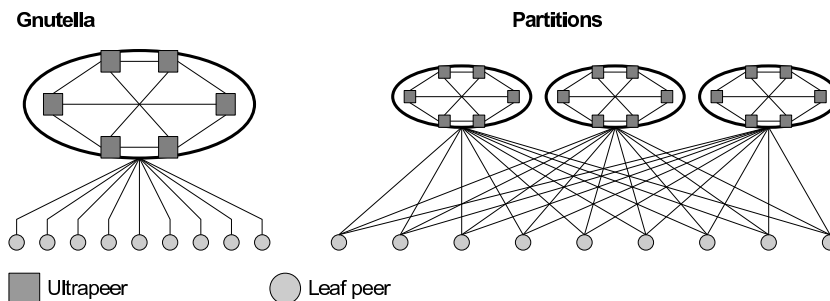


Figure 3: Illustration of the Gnutella network and the Partitions design.

4.1 Hash-based overlay partitioning

Partitions employ a universal uniform hash function to map each keyword to an integer, from a small set of integers. Each integer defines a different category. Thus, keywords are categorized instead of services/content. The keyword categories are exploited in a 2-tier architecture, where nodes operate as Ultrapeers and/or Leaves. More specifically, the partitioning of the network is performed as follows (see Fig. 3):

- Each Ultrapeer is randomly and uniformly assigned responsibility for a single keyword category. Ultrapeers responsible for the same category form a random subnetwork. As a consequence, the network overlay is partitioned into a small number of distinct subnetworks, equal to the number of available categories.
- Leaves randomly connect to one Ultrapeer per subnetwork. Furthermore, each Leaf sends to each Ultrapeer it is connected to all its keywords, in the form of a bloom filter, that belong to that same category. Thus, an innovative index splitting technique is used. Instead of each Leaf sending its entire index (keywords) to each Ultrapeer it is connected to, each Leaf splits its index based on the defined categories and constructs a different bloom filter for each keyword category. Each bloom filter is then sent to the appropriate Ultrapeer. An illustration of this technique can be found in Fig. 4.

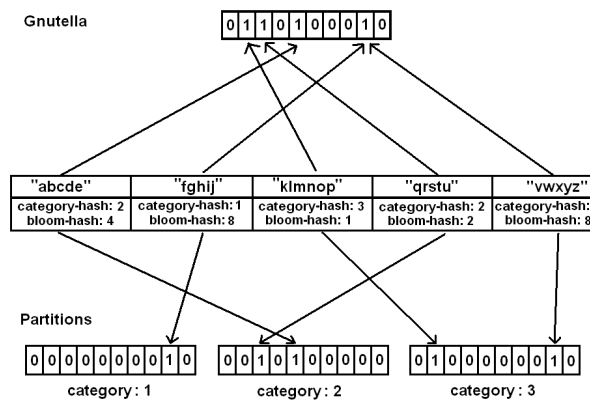


Figure 4: The Partitions and Gnutella bloom filters.

We should emphasize that in this design Ultrapeers are de-coupled from their content, meaning that peers operating as Ultrapeers will have to operate as Leaves at the same time in order to share their own content, which spans several categories. Furthermore, even though in this design each Leaf connects to more than one Ultrapeers, the volume of information it collectively transmits to all of them is roughly the same since each part of its index is sent to a single Ultrapeer.

The Partitions scheme is demonstrated in Fig. 3. The unstructured overlay network is partitioned into distinct subnetworks, one per defined category. A search for a keyword of a certain category will only flood the appropriate subnetwork and avoid contacting Ultrapeers in any other network partition. The benefit of this is two-fold. First, it reduces the size of the search for each individual request. Secondly, it allows each Ultrapeer to use all its Ultrapeer connections to connect to other Ultrapeers in the same network partition, increasing the efficiency of 1-hop replication at the Ultrapeer level. One-hop replication dictates that each Ultrapeer contains an index of the contents of its neighbouring Ultrapeers (including the contents of their Leaves).

There are, however, two obvious drawbacks to this design. The first one is due to the fact that each Leaf connects to more than one Ultrapeers, one per content category. Even though each Leaf sends the same amount of index data to the Ultrapeers collectively upon connection as before, it requires more keepalive messages to ensure that its Ultrapeer connections are still active. Keepalive messages however are very small compared to the average Gnutella protocol message. In addition, query traffic is used to indicate liveness most of the time, thus avoiding the need for keepalive messages.

The second drawback arises from the fact that each subnetwork contains information for a specific keyword category. Requests however may contain more than one keywords and each result should match all of them. Since each Ultrapeer is aware of all keywords of its Leaves that belong to a specific category, it may forward a request to some

Leaf that contains one of the keywords but not all of them. This fact reduces the efficiency of the 1-hop replication at the Ultrapeer level and at the Ultrapeer to Leaf query propagation. This drawback is balanced in two ways. The first is that even though the filtering is performed using one keyword only, Leaves' bloom filters contain keywords of one category, which makes them more sparse, thus reducing the probability of a false positive. Furthermore, the most rare keyword can be used to direct the search, further increasing the effectiveness of the search method.

4.2 Simulation results

We present the results from the simulations we conducted, in order to measure both the efficiency of the Partitions scheme in terms of cost of flooding (in messages) and maintenance cost (keepalive messages). The first metric measures the traffic load of the flooding process in the entire network, while the second metric focuses on the load experienced by a(ny) single Ultrapeer. We performed several simulations, varying the number of Leaves per Ultrapeer and the number of categories (partitions). In addition, apart from the Gnutella and the Partitions scheme, we also conducted simulations on a modified version of the Partitions scheme, called *Replication*. The only difference between Partitions and Replication is the fact that Leaves send a complete (containing all keywords, regardless of category) bloom filter to all the Ultrapeers they connect to.

In all simulations, we assumed a Leaf population of 2 million, a number reported by LimeWire Inc [1]. Each peer contains a number of files (and hence keywords) derived from a distribution obtained from real-world measurements [6]. Each Ultrapeer in the Gnutella network serves, on average, 30 Leaves, a number also obtained from real-world measurements [7]. In addition, we performed simulations for 10 and 60 Leaves per Ultrapeer. Each Ultrapeer in the Partitions design serves a number of Leaves equal to the number of categories (partitions) multiplied by the number of Leaves per Gnutella Ultrapeer. Since each Leaf sends to each Ultrapeer a fraction of its keywords (namely, a fraction equal to the number of categories), this results in a roughly similar number of keywords in a Partitions Ultrapeer as in a Gnutella Ultrapeer. The number appearing next to each scheme name in the legend of the graphs denotes the number of full indices stored in each Ultrapeer. For instance, Gnutella 30 is the classic Gnutella algorithm (each Ultrapeer serves 30 Leaves). In Partitions 10, each Ultrapeer serves 100 Leaves, receiving one-tenth of each one's index, thus adding up to 10 full indices. In all schemes, each Ultrapeer has thirty Ultrapeer-neighbours. Finally, we have used a Zipf-like distribution for the popularity of keywords, both in the peers' filenames and in the queries. The end-result is the one reported in [7], with Leaves having, on average, a 3% full bloom filter and a Gnutella Ultrapeer with 30 Leaves having a 65% full bloom filter. The size of the bloom filter array is the same as the one used in Gnutella today, which is 2 to the power of 16 (65536). The percentage of fullness of the bloom filter is the probability that an one-keyword query will be forwarded to a Leaf or a last-hop Ultrapeer.

Fig. 5(a) shows the efficiency of the Partitions scheme in reducing the cost of the flood. We can see that the drawback of filtering using only one keyword is balanced by the fact that the sparser Leaf indices (since they contain only one keyword category) produce less false positives, but mainly outweighed by the message reduction due to the partitioning of the network and therefore the reduction of the search space. In addition, it follows from the results that the benefits of being able to filter using all keywords in a query when forwarding to the Leaf layer (Replication scheme) is small compared to the increased maintenance cost. We would like to emphasize that each Partitions bloom filter (i.e. containing keywords of a certain category) has the length of a Gnutella bloom filter. Thus, one can roughly think of all the bloom filters of a single Partitions leaf as a (distributed) Gnutella bloom filter of 10 times the length (due to the 10 different categories). However the bandwidth needed to transfer such a bloom filter is not 10 times that of a Gnutella bloom filter, mainly because sparser bloom filters are compressed more efficiently.

Another interesting observation is the fact that Gnutella 10 outperforms Gnutella 30, even though the Ultrapeer layer is 3 times bigger. This can be explained by the fact that the Ultrapeers' bloom filters are more empty (since they contain the aggregate bloom filters of only 10 Leaves instead of 30). This means that the filtering during the last hop of flooding is more accurate, increasing the efficiency of 1-hop replication.

We then focus on the traffic load experienced by a single Ultrapeer. In all cases we simulated three hours in the life of a single Ultrapeer, with Leaves coming and going. Each time a Leaf is connecting to the Ultrapeer, it sends its index information, which is propagated by the Ultrapeer to its thirty Ultrapeer neighbors. In addition, we assumed that, periodically (every 10 seconds), each Ultrapeer receives a small keep-alive message from each Leaf and replies with a similar message to each one of them, unless a query and a reply were exchanged during the specified period. For each communication taking place, we measured the incoming or outgoing traffic in bytes, in order to estimate the bandwidth requirements. We used a keep-alive message size of 50 bytes and a query message size of 80 bytes, as indicated by the Gnutella Protocol Specification. In addition, for every 1400 bytes for each message sent, we added

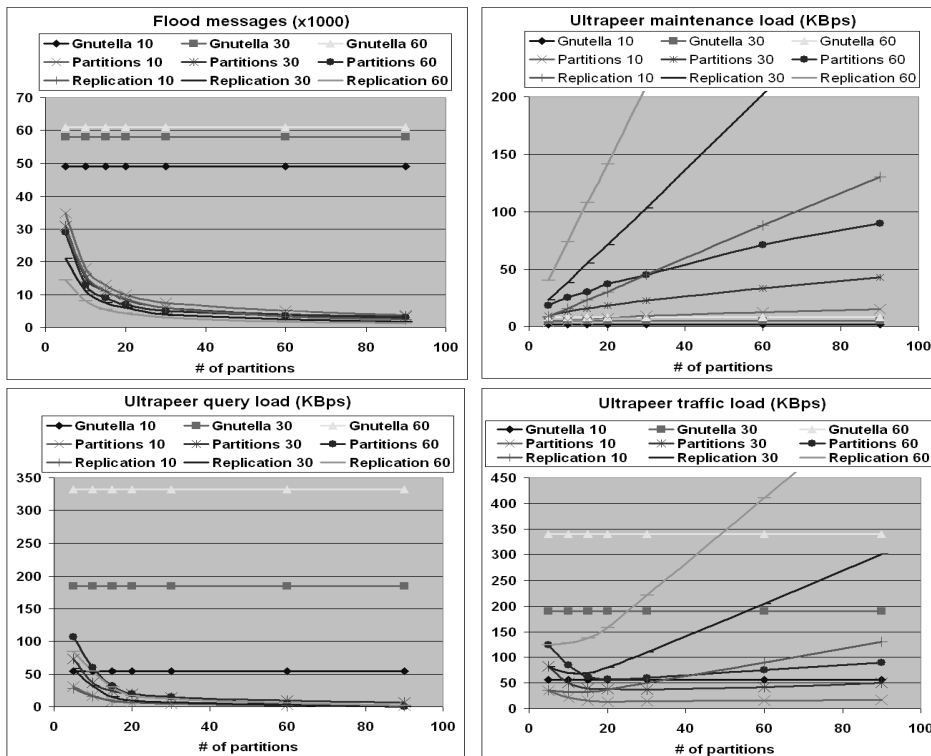


Figure 5: (a) Number of messages generated in one flood. (b) Maintenance traffic load. (c) Query traffic load. (d) Operational traffic load.

40 bytes for the TCP and IP header.

We have used the code by LimeWire [1], the most popular Gnutella client, to construct the bloom filter of each Leaf. For each peer, we then extracted a number of files (equal to the number of files assigned to that peer) from a list of filenames obtained from the network by a Gnutella crawler developed in our lab. Those filenames were fed to the LimeWire bloom filter generation code, which produced the corresponding bloom filter in compressed form, i.e., the way it is sent over the network by LimeWire servers. Thus, we constructed the actual bloom filter, although what we really needed was just its size (and the ratio of "fullness").

We run three types of simulations measuring (1) the maintenance traffic (bloom filter and keep-alive messages exchange), (2) the query traffic and (3) the total traffic. In the last case, we observed the nearly total lack of keep-alive messages, due to the implicit use of query traffic for indication of liveness. In all the graphs presented, the x axis corresponds to the number of partitions (keyword categories).

Fig. 5(b) shows the increase in the maintenance load when employing our scheme. As expected, the maintenance load in the Replications scheme increases linearly with the number of available partitions. This is because, the number of Leaves each Ultrapeer has to serve also increases linearly with the number of partitions. We can see that this is not the case with the Partitions scheme, where, even though the number of Leaves per Ultrapeer also increases, the size of the index submitted by each Leaf decreases.

We then focused our attention to the query traffic load. Measurements conducted in our lab showed that, on the average, each Ultrapeer generates 36 queries per hour (i.e., queries initiated by itself or its Leaves). This adds up to approximately 2000 queries per second generated anywhere in the Gnutella network. In addition, we observed a large number of Gnutella queries in order to find the distribution of the number of keywords in each query. Thus, according to those observations, during the simulations we assumed that 20% of the queries contain 1 keyword, 30% contain two, another 20% contain three and finally a 30% contain 4 keywords. In our simulation, we assumed that the aim of each flood (both in Gnutella and Partitions) is to reach the entire network, or produce a fixed number of results, whichever is achieved first. A flood that aims to reach the entire network would need to reach $\frac{1}{10}$ th of the Gnutella's network (or a Partitions' subnetwork) during all hops of flooding except the last thanks to the 1-hop replication. This

means that the Ultrapeer in our simulations has a probability of 0.1 to receiving each query. In addition, every time this does not occur, it has another opportunity to receive the query during the last hop, depending on its bloom filter (in case the searched keywords match in the bloom filter). Should the Ultrapeer receive a query, it is assumed to propagate it to its Leaves, again depending on their bloom filters. Fig. 5(c) shows the comparison in the traffic load of Gnutella, Partitions and Replication. Finally, fig. 5(d) shows the total traffic load experienced by a single Ultrapeer. From these figures it is evident that Partitions outperform Gnutella in operational costs, in all cases.

5 Ant-inspired reorganization of descriptors

Inside each partition, the construction of a composite service (or “workflow”) needs the identification of the basic services that will compose the workflow, and the discovery of such services on the network. This is generally reduced to the problem of finding *service descriptors*, through which it is possible to access the corresponding services.

In general, the construction of a workflow implies the generation of a discovery request for each of the required basic services, which can result in long discovery times and high network loads. The technique described in Section 4 allows for reducing flooding generated by discovery operations (by creating different domains). To further enhance performances, it would be useful to place descriptors of services that are often used together (i.e., in the same composite services) in restricted regions of the system, so that a single discovery operation will have high chances to find all or most of the required services in a short time (groups). Accordingly, we propose an ant-inspired technique to reorganize and sort the service descriptors in order to facilitate the composition of workflows.

Descriptors are indexed through bit strings, or *keys*, that are generated by a hash function. The hash function is assumed to be locality preserving [3, 8], which assures that similar descriptor keys are associated to similar services. In this context, the concept of similarity is defined in an ad hoc fashion: two services are defined as *similar* if they are often used together to compose a workflow. This type of similarity must be based on a statistical analysis of co-occurrences of services in the workflows characterized by similar semantics.

Our algorithm is inspired by the behavior of some species of ants [2], that sort items or corpses within their environment. The algorithm described in this report has been specifically designed to *sort* service descriptors, i.e., to place descriptors of services that are often co-used in composite services in neighbor peers, in order to facilitate and speed up their discovery. This work is inspired by the work of Lumer and Faieta [5], who devised a method to spatially sort data items through the operations of robots. In our case, descriptors are not only sorted, but also *replicated*, in order to disseminate useful information on the system and facilitate discovery requests. The behaviour of ants is here imitated by mobile agents that, while hopping from one peer to another, can copy and move service descriptors. Agents are able to disseminate and sort descriptors on the basis of their keys. Therefore, services descriptors individuated by similar keys will likely be placed in neighbour peers. This facilitates the composition of workflows in three ways:

- the ant-inspired agents are able to create and disseminate replicas of service descriptors, thus giving discovery operations more chances to succeed and improving the fault tolerance characteristics of the system;
- the discovery of a single service is facilitated because discovery messages can be driven towards the target descriptor in a very simple way. At each step the discovery message is sent to the neighbor peer whose descriptors are the most similar to the target descriptor specified in the query. Due to the sorting of descriptors in most cases this method allows queries to reach peers that store a significant number of useful descriptors;
- once a target descriptor has been reached, it is possible to locate other services, needed in the same workflow, in the same network region. Indeed, since services that are often co-used have similar keys, they have likely been placed into very close peers by the mobile agents of the ant algorithm.

5.1 Ant-inspired algorithm

In this subsections we briefly describe the ant-inspired algorithm, but more details can be found on [4]. Periodically each agent performs a small number of P2P hops among Grid hosts. When an agent arrives at a new Grid host, if it is carrying some descriptors it must decide whether or not to *drop* these descriptors whereas, if it is currently unloaded, it must decide whether or not to *pick* one or more descriptors from the local host. When performing a pick operation, an agent must also decide if the descriptor should be replicated or not. In the first case, the descriptor is left on the current peer and the agent will carry a replica; in the other case, the descriptor is taken from the peer and carried by the agent. This way, agents are able to replicate, move and reorganize the descriptors.

In both cases, the agent's decisions are based on a similarity function, f , which is reported in formula (1) and is based on the basic ant algorithm reported in [5]. This function measures the average similarity of a given descriptor \bar{d} with all the other descriptors d located in the *local region* R .

For each peer p , the local region includes all the hosts that are reachable from p with a given number of hops. This number is an algorithm parameter, and is set here to 1, in order to limit the amount of information exchanged among hosts.

In formula (1), N_d is the overall number of descriptors maintained in the region R , while $H(d, \bar{d})$ is the Hamming distance between d and \bar{d} . The parameter α defines the similarity scale; here it is set to $B/2$, which is half the value of the maximum Hamming distance between vectors having B bits. The value of f assumes values ranging between -1 and 1, but negative values are truncated to 0.

$$f(\bar{d}, R) = \frac{1}{N_d} \cdot \sum_{d \in R} \left(1 - \frac{H(d, \bar{d})}{\alpha}\right) \quad (1)$$

The probability of picking a descriptor stored in a peer must be inversely proportional to the similarity function f , thus obtaining the effect of averting a descriptor from co-located dissimilar descriptors. Conversely, the probability of dropping a descriptor carried by an agent must be directly proportional to the similarity function f , thus facilitating the accumulation of similar descriptors in the same local region.

The pick and drop probability functions, $Ppick$ and $Pdrop$, are defined in formulas (2)a and (2)b. In this functions, the parameters k_p and k_d , whose values are comprised between 0 and 1, can be tuned to modulate the degree of similarity among descriptors.

$${}^{(a)} Ppick = \left(\frac{k_p}{k_p + f}\right)^2 \quad {}^{(b)} Pdrop = \left(\frac{f}{k_d + f}\right)^2 \quad (2)$$

After evaluating the pick or drop probability function (which function depends whether the agent is carrying descriptors or not), the agent computes a random number comprised between 0 and 1 and, if this number is lower than the value of the corresponding function, it executes the pick or drop operation for the descriptor under examination. The inverse and direct proportionality with respect to the similarity function f assures that, as soon as the possible initial equilibrium is broken (i.e., descriptors having different keys begin to be accumulated in different Grid regions), the reorganization of descriptors is more and more facilitated.

The effectiveness of the algorithm is evaluated through the spatial *homogeneity function* H . Specifically, for each peer p of the Grid, the average homogeneity H_p of the descriptors located in the local region of p , R , is calculated. This is obtained, as shown in formula (3), by averaging the Hamming distance between every couple of descriptors in R and then subtracting the obtained value from B , which is equal to the maximum Hamming distance. Thereafter, the value of H_p is averaged over the whole Grid, as formalized in formula (4).

$$H_p = B - AVG_{\{d_1, d_2 \in R\}}(H(d_1, d_2)) \quad (3)$$

$$H = \frac{1}{N_p} \cdot \sum_{p \in Grid} H_p \quad (4)$$

Our objective is to increase the homogeneity function as much as possible, because it would mean that similar descriptors are actually mapped and aggregated into neighbor hosts, and therefore an effective sorting of descriptors is achieved.

Figure 6 reports the values of the overall homogeneity function, H , for different values of the number of bits. It confirms that the work of agents makes this index increase from about $B/2$ to much higher values. It is interesting to note that the increase in H hardly depends on the value of B , which means that the algorithm is able to reorganize descriptors regardless of the accuracy with which resources are actually described.

6 Conclusions

The report proposes an innovative approach to the distributed and cooperative composition of Grid (and not only) services based on: (1) AI planning techniques, (2) service grouping and (3) overlays in large P2P networks. To this

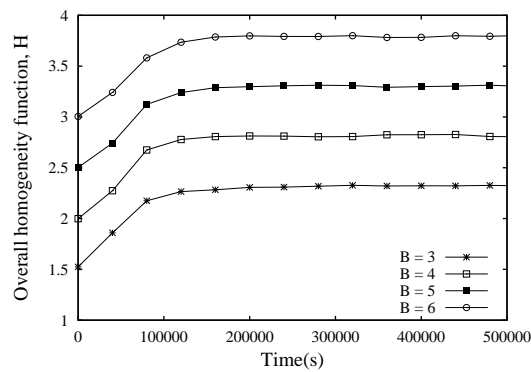


Figure 6: Overall homogeneity function vs. time, for different values of the number of bits in resource descriptors, B .

end, we integrated in a system for distributed service composition an intelligent, ant-inspired search mechanism able to self-organize the location of the services to facilitate discovery with a partitioning technique that reduces flooding. The end system is a composite mechanism that can scale to a large number of services. So, future work is aimed at improving the integration by testing the results in both simulated and real environments.

References

- [1] Limewire Inc. <http://www.limewire.com>
- [2] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, New York, NY, USA, 1999.
- [3] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. Maan: A multi-attribute addressable network for Grid information services. In *GRID '03: Proceedings of the Fourth International Workshop on Grid Computing*, page 184, Washington, DC, USA, 2003. IEEE Computer Society.
- [4] Agostino Forestiero, Carlo Mastroianni, and Giandomenico 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*, Budapest, Hungary, December 2007.
- [5] Erik D. Lumer and Baldo Faieta. Diversity and adaptation in populations of clustering ants. In *Proc. of SAB94, 3rd international conference on Simulation of adaptive behavior: from animals to animats 3*, pages 501–508, Cambridge, MA, USA, 1994. MIT Press.
- [6] R. Rejaie, Shanyu Zhao, D. Stutzbach. Characterizing files in the modern Gnutella network: A measurement study. In *Proc. SPIE/ACM Multimedia Computing and Networking*, 2006.
- [7] D. Stutzbach and R. Rejaie. Characterizing the two-tier gnutella topology. In *Proc. of the ACM SIGMETRICS*, 2005.
- [8] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *Proc. of the 14th IEEE International Symposium on High Performance Distributed Computing HPDC 2005*, Research Triangle Park, NC, USA, July 2005.
- [9] Harris Papadakis, Paraskevi Fragopoulou, Marios Dikaiakos, Alexandros Labrinidis and Evangelos Markatos. *Divide Et Impera: Partitioning Unstructured Peer-to-Peer Systems to Improve Resource Location*. CoreGRID Springer Volume, 2007.
- [10] Alberto Troisi, Eugenio Zimeo. Self-Organizing Service Network in a P2P environment. *Technical Report*, Research Centre on Software Technology - University of Sannio, Italy, 2007.
- [11] ebXML. ebXML: electronic business using extensible markup language. <http://www.ebxml.org>

- [12] UDDI 3.0 Universal description, discovery and integration version 3. <http://www.uddi.org>
- [13] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, John Miller. *METE-ORDS WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services*. Information Technology Management, (6)1:17-39, 2005.
- [14] T.Pilioura, G. Kapos, and A. Tsalgatidou. PYRAMID-S: a scalable infrastructure for semantic web services publication and discovery. In *Proc. of the 14th International Workshop on Research Issues on Data Engineering*, 28-29 March 2004.
- [15] Luciano Baresi, Matteo Miraz. A Distributed Approach for the Federation of Heterogeneous Registries. In *Proc. of ICSOC 2006*, Chicago, USA, 2006.