A P2P Resource Discovery System Based on a Forest of Trees*

Moreno Marzolla[2] Matteo Mordacchini[1,2] Salvatore Orlando[1,3]
[1] Dip. di Informatica, Università Ca' Foscari di Venezia, via Torino 155, 30172 Mestre, Italy
[2] INFN Sezione di Padova, via Marzolo 8, 35100 Padova, Italy
[3] ISTI Area della Ricerca CNR, via G. Moruzzi 1, 56124 Pisa, Italy
{moreno.marzolla|matteo.mordacchini}@pd.infn.it, orlando@dsi.unive.it

Abstract

The convergence of the Grid and Peer-to-Peer (P2P) worlds has led to many solutions that try to efficiently solve the problem of resource discovery on Grids. Some of these solutions are extensions of P2P DHT-based networks. We believe that these systems are not flexible enough in the case the data present in the network is very dynamic, i.e. the data values change very frequently over time. This a very common case for some kind of data in typical Grid systems, like CPU loads, queue occupation, etc. In this paper we present a variation of a previous work based on a tree-shaped P2P overlay network. The system uses Routing Indexes to efficiently route queries and update messages in the presence of highly variable data. The new system is based on a two-level hierarchical network topology, where tree topologies must be only maintained at the lower level of the hierarchy. The main aim of this new organization is to achieve a simpler maintenance of the overall P2P graph topology. This is reached at the cost of some imprecision in the routing indexes built at the upper level of the graph hierarchy.

1. Introduction and Related Works

The convergence of Grid and P2P systems has led to the development of many solutions that try to combine the advantages of the two worlds. One of the fields in which this convergence seems to give the most promising results is resource discovery. Many solutions described in the literature are based on Distributed Hash Tables (DHT), like [9], [7] and [8]. These systems organize the peers into overlay networks and create distributed indexes associated with the resources shared among the network, in order to let users to efficiently retrieve them. The above-mentioned systems allow exact queries only, while one of the common way to retrieve data on Grids is through multi-attribute range queries. Some authors have proposed extension of the cited algorithms in order to adapt these DHT systems to the Grid needs [3] [1] [2]. Despite the good results achieved by this kind of networks, they may not represent the best solution in presence of dynamic data, i.e. data whose value change frequently and unpredictably over time. The main problem with DHT-based networks is related to the fact that each change in the resource values requires to re-index the items whose content have changed.

We believe that less structured networks are more suitable to deal with dynamic data. In a previous work [6] we proposed a system based on Routing Indexes (RI) [4] in order to find a good tradeoff between an efficient query routing scheme and a good update mechanism. In this paper we present a variation of that system based on a new topology organization of the peers. This new organization aims to ease the maintenance of the network while preserving the good features of the old topology organization, based on a tree-shaped P2P overlay network. To avoid the cost and difficulties in maintaining a very large tree topology, in this paper we propose a two-level hierarchical network topology, where tree topologies must be only maintained at the lower level of the hierarchy. The upper level is instead unstructured. This fact may introduce some imprecision in the indexing system. Our idea is to tradeoff a simpler maintenance of the overall P2P graph topology, with the introduction of some imprecision in the routing indexes built at the upper level of the graph hierarchy.

The rest of the paper is organized as follows: in Section 2 we give a detailed description of our system; in Section 3 we present some simulation results showing the performance of the proposed algorithms; finally, conclusions and future works are illustrated in Section 4

*This work has been partially supported by the European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies (CoreGRID) and the EU Project EGEE (Enabling Grids for E-sciencE)

2. System Overview

We first describe the assumptions we make through the rest of the paper. We consider a Grid system composed of N Resource Brokers (RB). Each RB holds a set of resources, called the *local repository*. Each resource is characterized by a set of attributes. For each resource R, we denote with $\mathit{AttList}(R)$ the set of all attribute names defined for R. For each attribute $A \in \mathit{AttList}(R)$, $R[A]$ denotes the value of attribute A for resource R. As an example if we consider a resource of type "Computing Element", then attributes for such resource may be "number of processors", "available memory", "number of queued jobs" and so on. We suppose that resources are dynamic, in the sense that the value of the attributes may change unpredictably over time. Users of the Grid need to query the system in order to locate the most suitable resources that are required to execute their jobs. We assume that users specify their needs by means of partial range queries generated by the following grammar (we assume that the usual operator precedence rules apply):

$$\begin{aligned} Q &:= Q \textbf{ and } Q \mid Q \textbf{ or } Q \mid v_1 \leq A \leq v_2 \\ A &:= A_1 \mid \ldots \mid A_M \end{aligned}$$

We consider partial range queries over subsets of the attributes, that is, boolean compositions of range predicates $v_1 \leq A \leq v_2$. Multiple conditions over different attributes are possible. Conditions such as $A \leq v_2$, $A \geq v_1$ and $A = v_1$ are special cases of $v_1 \leq A \leq v_2$ which can be expressed by setting $v_1 = -\infty$, $v_2 = +\infty$ and $v_1 = v_2$ respectively.

In order to find a good tradeoff between an efficient search mechanism and the need for dealing with dynamic data values, we proposed in [6] to organize the RBs into a P2P tree-structured overlay network, with routing information associated with links in the network. Despite its good performance properties, this structure has two main drawbacks. First, the topology may become hard to maintain when nodes join and leave the network, especially in the case we want the tree to remain (almost) balanced. Second, the nodes close to the tree root may become overloaded of routing requests. For these reasons, in this paper we propose to partition the network into a set of small trees, i.e. a forest. Each peer P belongs to a single tree, which we call *group* of P. The nodes that have a link with P are of two types: (1) nodes of same group, i.e. *local neighbors*, denoted with $\mathit{LNb}(P)$; (2) nodes that belong to other groups, i.e. *external neighbors*, denoted with $\mathit{ENb}(P)$. The set of all the peers connected with P is called the *neighborhood* of P and is denoted with $\mathit{Nb}(P)$, i.e. $\mathit{Nb}(P) = \mathit{LNb}(P) \cup \mathit{ENb}(P)$. In order to have a connected network we impose that for each group of nodes, at least a node of the group must have an external neighbor. An example of a network of this type is shown in Fig. 1. For sake of simplicity, the figure presents a *vertex clustering* of the network, i.e. only connections between groups are drawn.

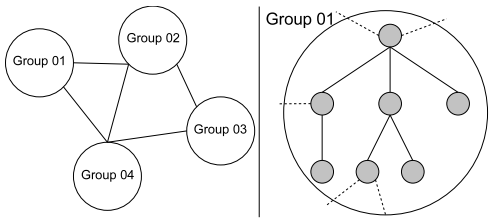


Figure 1. An example of the new network organization

We now show how routing information are associated with links between peers and how the network structure is exploited to manage query routing requests and update messages.

2.1. Bitmap Indexes

Each peer P maintains a summary of all informations of its local resources as follows. If the domain of attribute A is the interval $[a, b]$, we select $k+1$ division points $a = a_0 < a_1 < \ldots < a_k = b$ such that $[a, b]$ is partitioned into k disjoint intervals $[a_i, a_{i+1})$, $i = 0, 1, \ldots k-1$. Given an attribute A we encode the value $D[A]$ with a k bit binary vector $\mathit{BitIdx}(D[A]) = (b_0, b_1, \ldots, b_{k-1})$, such that $b_i = 1$ if and only if $D[A] \in [a_i, a_{i+1})$. Both the parameter k (number of bits of the bit vector) and the division points $a_0, a_1, \ldots a_k$ may be different for each attribute type. The final local bitmap index for an attribute A on peer P is obtained by a bitwise OR operation between the indexes of all data items D of P:

$$\mathit{BitIdx}(P, A) \equiv \bigvee_{D \in \mathit{Data}(P)} \mathit{BitIdx}(D[A]) \qquad (1)$$

where $\mathit{Data}(P)$ is the set of all the resources of P. Each peer P_i also maintains a condensed representation of the resources which can be found by following its outgoing links. In particular, given $P_j \in \mathit{Nb}(P_i)$, let G_j be the group of P_j and $\mathcal{T}_{G_j}$ be the internal tree of group G_j. Let $\mathcal{T}_{G_j}(P_i \rightarrow P_j)$ be a subtree of $\mathcal{T}_{G_j}$ rooted on P_j such that $\mathcal{T}_{G_j}(P_i \rightarrow P_j) \subseteq \mathcal{T}_{G_j} \backslash \{P_i\}$. For each data attribute A, P_i associates with the edge (P_i, P_j) the following quantity:

$$\mathit{LinkBitIdx}(P_i \rightarrow P_j, A) \equiv \bigvee_{P \in \mathcal{T}_{G_j}(P_i \rightarrow P_j)} \mathit{BitIdx}(P, A) \qquad (2)$$

Note that, at this level, we do not need to make distinctions between internal or external links. The result of this operation is that the $\mathit{LinkBitIdx}(\cdot)$ index gives to P_i a description of the data items that can be found in the portion of the

group tree rooted on neighbor P_j, if P_j is in the same group of P_i, or a description of the entire set of resources in the external group G_j.

2.2. Query Routing

We assume that queries originate from any node P in the system. Upon receiving a query message, a node checks if some of its resources match the query; then, it forwards the query to its neighbors. Queries are not necessarily routed to all neighbors but only to a selected set of them. The selection phase is driven by the routing indexes associated with each outgoing link. According to Eq.(2), each peer P maintains a bit vector $\mathit{LinkBitIdx}\,(P \rightarrow P', A)$, for each node $P' \in \mathit{Nb}\,(P)$. When P receives a query $Q := v_1 \leq A \leq v_2$ from a neighbor P_{in}, it forwards it to neighbor $P_{out} \in \mathit{Nb}\,(P) - P_{in}$ only if a match is likely to be present in $\mathcal{T}(P \rightarrow P_{out})$. To this end, the result of the logical AND between the bitmap representation of the query range and $\mathit{LinkBitIdx}\,(P \rightarrow P_{out}, A)$ must be a non-zero vector. The routing scheme described so far may not be enough for an efficient routing of queries. Note that the outer graph, associated with the inter-group connections, is unstructured. Therefore, it may also include loops among the (super) nodes. Let us consider the situations depicted in Fig. 2. For the sake of simplicity, only links between the presented nodes are shown. In the first case, P_1, P_2 and P_3 have at least a resource matching uery Q .Let us suppose the P_1 receives Q. After P_1 has detected the local match, it forwards the query to its neighbors. Node P_2 forwards the query to P_3. P_3 will then send the query back to P_1, that, in turn, will forward it again to P_2. The final result is an infinite loop. In order to avoid loops, we need to improve

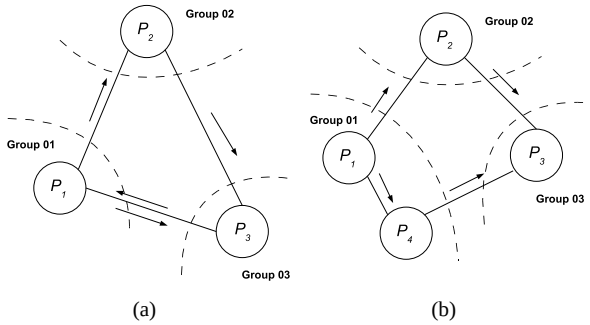


Figure 2. Examples of loops (a) and multiple resolutions of the same query (b)

the routing scheme. We add a further information to the query message, as shown in Fig. 3. We add the list of all the IDs that are associated with the groups of nodes that were already traversed by the query. Let G_Q be such a list. A peer P that receives query Q adds its own group ID to G_Q,

Query ID	Query Upper Bound	Query Lower Bound	Query Bitmap Representation B_q

Figure 3. Information associated with a query

if it is not already present. Then it forwards Q to the external neighbors whose groups are not listed in G_Q. Nothing changes for the internal neighbors. This solution eliminate cycles but does not completely eliminate the problem illustrated in Fig. 2(b). A query Q is forwarded by node P_1 to all nodes P_2, P_3 and P_4 that match it. As can be seen, node P_3 receives the same query from node P_2 and node P_4. The previously described mechanism avoids the formation of a loop but does not prevent P_3 from processing the same query twice. For this reason we need to add to each peer P a *query cache*, i.e. a cache that contains the latest *m* query IDs received by P. If P receives a query whose ID is already listed in its own query cache, it simply discards the query, avoiding to process it twice.

2.3. Handling Updates

In this section we describe how the system deals with updates. Suppose that peer P notes a change in $D[A]$ for a data item $D \in \mathit{Data}\,(P)$. Let v_{new} and v_{old} be the new and old values of $D[A]$, respectively. The first action taken by P is to compute the bitmap representation of v_{new}, $\mathit{BitIdx}\,(v_{new})$. If it is equal to $\mathit{BitIdx}\,(v_{old})$, no other actions are needed. Otherwise, P computes the new $\mathit{BitIdx}\,(P, A)$. Again, it may happen that the new index is the same as the old one, and then no further actions are required. If the new index differs from the previous one, an update message is propagated in order to preserve the property defined by Eq. 2. Update messages consists of the name of the attribute whose value is changed, and its up-to-date bit vector representation. The new bit vector is computed as follows:

$$\mathit{LinkBitIdx}\,(P_{out} \rightarrow P, A) = \mathit{BitIdx}\,(P, A) \vee \left(\bigvee_{P' \in \mathit{LNb}(P) - P_{out}} \mathit{LinkBitIdx}\,(P \rightarrow P', A) \right) \quad (3)$$

If $P_{out} \in \mathit{ENb}\,(P)$, the index is computed over the all set of P local neighbors and thus it represents a description of all P's group resources. When a peer receives an update message from an incoming internal connection, it updates bit vector indexes according to Eq. 3 and sends them to internal and external neighbors, respectively. If a peer receives an update message from an external link, it simply registers the new index and does not perform any other action. It is worth remarking that the update messages are not further propagated once an external link is traversed.

3. Simulation

We performed simulation experiments in order to evaluate the performances of the proposed P2P system. We implemented a process-oriented simulation model of a set of interacting peers using the C++ library described in [5]. We analyze the *steady-state behavior* of the system using the *independent replication* approach to compute performance measures. For each simulation run we collect a fixed number (in our case, 200) of observations. The first 20% of the observations is discarded, in order to remove the initial transient. In the rest of this section we made the following assumptions. The term "group size" means the *maximum group size*, while the minimum group size is calculated as the 20% of the maximum group size. The resource density p indicates the probability of each node to posses a valid resource. Thus, the greater p, the higher the number of resources present in the network. Two nodes that belong to two different groups have a probability l to be connected. Such links determined the degree of connectivity of the groups of the network.

We now compare our previous algorithm and the new topology organization described in this paper. We first analyze the behavior of the system with respect to the query propagation phase. In particular, we consider the span of a query, i.e. the number of nodes receiving a query message, either because they matched the query or they simply forwarded it to their neighbors. The results are shown in Fig. 4. The simulation was performed considering three different

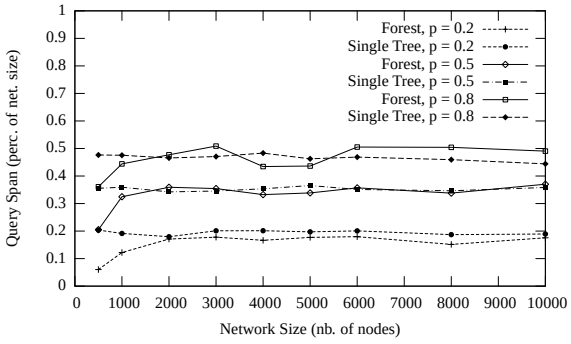


Figure 4. Query Span for the old and new algorithm, w.r.t. 3 different resource densities (lower is better)

resource densities and a network with groups of fixed size of 10 nodes. Moreover, we consider a random tree structure for both the single tree and the internal group trees. As we can see, the query span is proportional to the network size in both cases. This is due to the fact that the resources present in the network are also proportional to the number of nodes in the system, under our experimental hypothesis. The two algorithms have very similar performances, with the new one behaving slightly better for lower values of p. Other two important indicators are the precision and the recall of the two systems. In Fig. 5 we plot the precision achieved with a resource density $p = 0.5$ for both networks, a link rate $l = 0.01$ and a group size $g = 10$ for the new network. This latter solution performs slightly better than the old one. But, as said in the previous sections, some imprecision is introduced as a consequence of the new topology. The imprecision is due to the inability of a node to forward a query to some zones of the network where resources are available. This may be caused by an inadequate connection between groups in the network and a subsequent lack of information in the indexes. We analyzed the behavior of the new system when the inter-group link rate l varies. It does not influence the precision, since precision is more related to the nature of our indexing method: as indexes represent a condensed description of network resources, they influences the precision of the system, as stated in [6]. Thus, the most relevant effect of the inter-group link rate is on recall. Fig. 6 highlights this fact. As can be seen, the smaller the link probability, the worse the performances of the system. This is the result we expected, given the considerations above. We also analyze the update mechanism. In Fig. 7(a) the two systems are compared, and the new system behaves better. This is due to its group-based organization. In fact, the radius of an update is mainly determined by the group radius, as external nodes add only one more hop to the forwarding process, as illustrated in section 2.3. This is confirmed by the results presented in Fig. 7(b). The behavior of the system is analyzed with respect to three different resource densities. Resource density affects the results only when the group size change; in the old system the update radius depends mainly on the resource density.

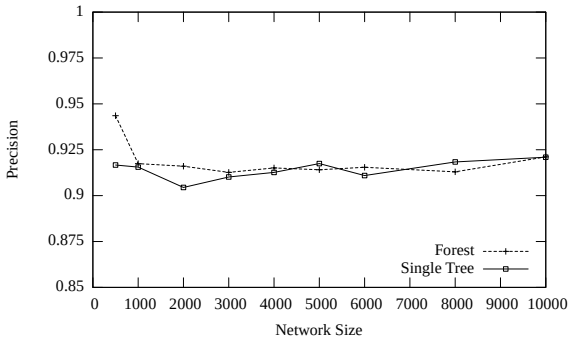


Figure 5. Precision of the new and old systems (upper is better)

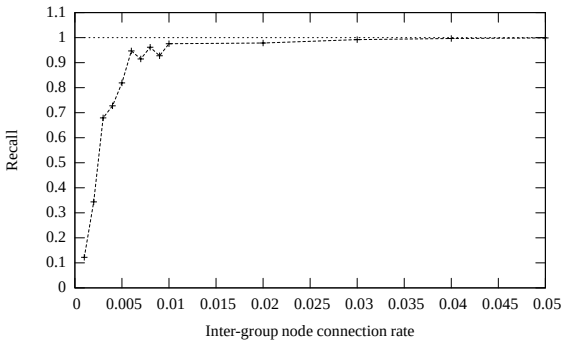
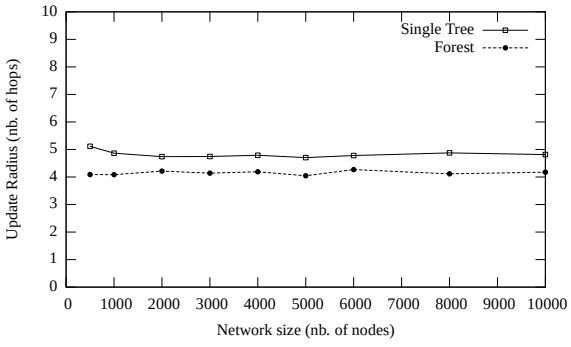
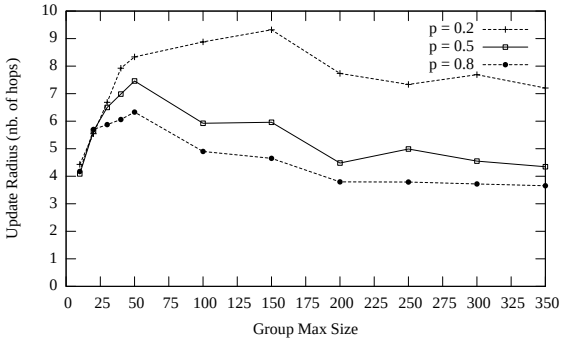


Figure 6. Recall for the new system, w.r.t. inter-group connection rate (upper is better)



(a)



(b)

Figure 7. Update Radius of the two systems w.r.t. network size (a) and of the new system w.r.t. groups size (b) (lower is better)

4. Conclusions and Future Works

In this paper we presented a P2P network for a Grid Information System designed to allow efficient discovery of resources with the use of multi-attribute, range queries and an effective handling of updates messages in the presence of dynamic data. The system is based on a the organization of peers within structured groups, while inter-group links follow a non-structured approach. Query routing and limitation of the flooding of updates are guaranteed by distributed indexes based on a bit vector representation of resource values. The simulation results show the effectiveness of the proposed approach.

We are currently anallyzing the behavior of the proposed algorithm with different kinds of resource distribution. Moreover, we plan to investigate the efficiency of the system when dealing with the *top-k* queries, instead of queries that try to retrieve all the matching resources.

References

[1] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *P2P '02: Proc. of the Second Int. Conf. on Peer-to-Peer Computing*, page 33, Washington, DC, USA, 2002. IEEE Computer Society.

[2] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proc. ACM SIGCOMM 2004 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 353–366. ACM Press, 2004.

[3] M. Cai, M. Frank, J. Chen, and P. Szekely. Maan: A multi-attribute addressable network for grid information services. In *GRID '03: Proc. of the 4th Int. Workshop on Grid Computing*, page 184, Washington, DC, USA, 2003. IEEE Computer Society.

[4] A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS '02: Proc. of the 22nd Int. Conf. on Distributed Computing Systems (ICDCS'02)*, pages 23–33, Washington, DC, USA, 2002. IEEE Computer Society.

[5] M. Marzolla. `libcppsim`: a simula-like, portable processoriented simulation library in c++. In *G. Horton, editor, Proc. ESM04, the 18th European Simulation Multiconference*, pages 222–227, Magdeburg, Germany, June 13-16, 2004. SCS Press.

[6] M. Marzolla, M. Mordacchini, and S. Orlando. Resource discovery in a dynamic grid environment. In *Proc. of DEXA Workshops 2005*, pages 356–360, Copenhagen, Danemark, August 22–26, 2005. IEEE Computer Society.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. SIGCOMM '01*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[8] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001: Proc. of the IFIP/ACM Int. Conf. on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[9] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.