

Distributed Grid Resource Discovery with Matchmakers

Mohammad Imran Shaik S. Mary Saira Bhanu Dr. N. P. Gopalan
National Institute of Technology, Tiruchirappalli, Tamilnadu, India - 620015
imran.sk@gmail.com msb@nitt.edu gopalan@nitt.edu

Abstract

The Grid provides mechanisms to share dynamic, heterogeneous, distributed resources spanned across multiple administrative domains. Resources required to execute a job are identified from the resource pool based on the desired set of attributes of the resources that exhibit various degrees of dynamism. A Resource Discovery service provides mechanisms to identify the set of resources that are capable of satisfying the job requirements. Current grid systems (Condor, European Data Grid) employ centralized matchmakers for resource discovery. Centralized matchmakers become a performance bottle neck as the grid grows. To alleviate this problem and to effectively exploit the grid, a distributed matchmaking approach is desirable not only for performance reasons, but also to surmount single point failures for grid resource discovery. In this paper, a novel distributed matchmaking approach for resource discovery in large, dynamically varying collections of grid resources is proposed.

1. Introduction

Resource discovery service in Grid is a basic service that identifies a set of suitable resources for a given job in dynamic, distributed, and heterogeneous environments. Given a description of job requirements, the service provides information (typically address) of resources that match the job requirements. In contemporary grid systems, resources are owned by research institutes, popular universities, and large organizations. They employ centralized resource discovery systems (e.g. Condor, Globus Toolkit). Advertisements of jobs and resources (Ads) are submitted to a central node that in turn finds suitable resource(s) and returns result(s) to the requester. The centralized resource discovery system becomes a performance bottleneck as the grid scales with increasing number of users and resources. To

address this problem a distributed resource discovery approach is desirable.

In this paper, we propose a distributed resource discovery mechanism with matchmakers. The matchmakers perform Condor [4] like Classified Advertisements (ClassAds) [5] based matchmaking. In this approach resource information is distributed across the grid. Local servers (we refer them matchmakers or nodes throughout the paper) provide information of the resources available locally. Users place requests for resources to their local matchmakers. Each organization or a group of organizations may own a matchmaker. A matchmaker is responsible for resource discovery within and outside the organization.

This paper is organized as follows. Section 2 deals with existing mechanisms for resource discovery, section 3 discusses the distributed grid resource discovery approach with request forwarding algorithm and section 4 shows experimental evaluation. Finally, section 5 concludes with future directions.

2. Related Work

A fully decentralized approach for grid resource discovery is discussed in [1]. Four request forwarding algorithms are evaluated for various resource usage and request patterns. Each algorithm attempts to find a resource for the given job request. None of the algorithms consider the query response time. Also, the algorithms do not make another attempt when a resource is not found in a single attempt. This negligence approach may end up finding no resource even when a resource is present in the grid.

A super-peer model proposed in [2] employs neighborhood search to find multiple resources for a given job. Hit count of a neighbor node is updated if a request forwarded via the neighbor returns results. A subsequent request propagates through the same adjacent neighbor. Only the neighbors learn in this approach, though they are not the exact providers of the resource. The algorithm does not discuss the

dynamic changes in the membership as it is based on predefined topology. The request message is annotated at every hop and becomes heavy as it advances in the grid. This approach may lead to poor response time in low bandwidth networks. Further, it doesn't take into account the average response time of a neighbor to forward a request.

Distributed resource discovery mechanisms used in Gnutella [7] and Napster [8] are based on filenames. Such naming strategies are inefficient in grids as grid resources are identified based on a set of desired attributes of the resources.

Current grids (e.g. Globus, Condor etc.) were originally designed to support centralized resource discovery mechanisms. The grid community accepts that a centralized approach is not feasible for future grids. Future grids outgrow organizational bounds in which case centralized discovery mechanisms become inefficient. Grid resource discovery mechanisms ought to be supplemented with P2P like algorithms for scalability and dynamism in grids.

In this paper, matchmakers perform ClassAds based matchmaking. They are distributed across the grid. Every resource is under control of a matchmaker. Each matchmaker attempts to satisfy the requests locally. If not, it forwards the request to a remote peer. Thus we employ powerful matchmakers locally and exploit the power of robust request forwarding algorithms at the global level.

3. Distributed Matchmaking Framework

In our framework, a grid is viewed as a collection of distributed matchmakers. Each organization or a group of organizations may own matchmaker(s). For simplicity we assume, each organization owns a matchmaker. Resource providers of an organization(s) make advertisements to the matchmaker. The matchmaker is responsible for finding suitable matches for the advertisements made to it. A matchmaker itself can act as a resource provider. Matchmakers communicate among themselves to discover non local resources. The actions of a matchmaker includes, (i) attempting to find a local resource for the given resource request, (ii) forward the request to a best neighbor matchmaker if no local match is found. By neighbor, we mean a peer node known to the matchmaker. Matchmakers keep track their active neighbors with periodic "I am alive" messages.

In this framework, we assume that all the nodes (matchmakers) have joined the grid and a resource can join and leave the grid at any time. So, matchmakers periodically collect and update the advertisements

information from the resources. Matchmakers do not guarantee to provide the current information, but ensures that the information is the latest update available with them. Further, the matched entities can initiate a claiming protocol to confirm the match information provided by the matchmaker.

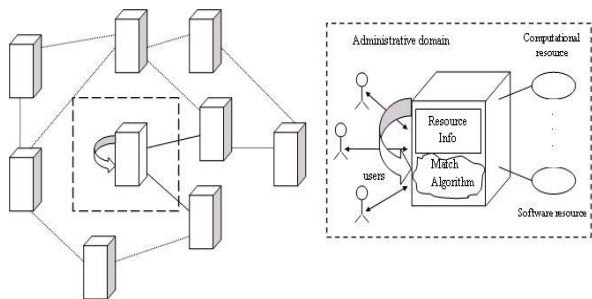


Figure 1. Visualization of the grid with distributed matchmakers

Distributed grid resource discovery is characterized by membership protocol and request forwarding algorithms. Each node maintains the list of neighbors in a neighbor list. The neighbor list is updated in the following scenarios; (i) a new node joins the grid (ii) a peer member node is known to the node (iii) a node merges with another node (iv) a node leaves the grid.

A node joins the grid by contacting a member node of the grid. Neighbor lists in either node are updated with the entries of the other. A node knows a peer member node when it receives results for a request from the latter. Both the membership lists are updated in this case too.

A node may wish to merge with another node when the node's resources are unutilized for a long period. It may try joining a frequently requested node, so that the chance of selling its resources is more. The merging node sends a *merge* request to the latter. After a successful negotiation with the requester node, the remote node confirms the merge by sending a *merge* command. The merging matchmaker then sends an asynchronous *neighbor update* message to all the neighbors indicating its new matchmaker. The merging node authenticates the remote node to access all of its resources. We consider this scenario different from a node leaving the grid, because the resources of the node are still available, but from a different matchmaker. Even without a "*neighbor update*" notice, our resource discovery algorithm performs well with the help of random forwarding technique embedded in it. By providing a notice, we wish to keep the other nodes informed.

The periodic "I am alive" messages exchanged between the nodes are used to keep track of the activity

of a node. A node is active if it is up and running. When a node leaves the grid, neighbors update their neighbor list by deleting the entry for the node.

Request forwarding is an important design decision that directly affects the success rate of resource discovery. Users send their requests to the local matchmaker. If a request is satisfied, match information (typically address) is returned to the entities involved, else the request is forwarded to a remote matchmaker. Matchmakers forward a request to a remote matchmaker when they cannot answer. A forwarding decision is made on the neighbor based upon the previous experience with the neighbor. If a match is found to a forwarded request, then the matchmaker sends the match information directly to the entity that initiated forwarding. If multiple matches are found at a remote matchmaker, the matchmaker constructs a single response message with the addresses of the all the matched resource providers. The request is removed when a match is found or its time to live value expires.

Each matchmaker caches the neighbor experience information in **experience-cache**. The experience cache consists of neighbor address, number of hits produced by the neighbor, the average response time taken by the neighbor to answer a query. A best neighbor is selected based on the above parameters.

3.1. Best Neighbor Algorithm

The best neighbor selection algorithm returns a neighbor with maximum number of query hits and minimum average response time. If no relevant experience information exists, the neighbor is randomly selected. If a forwarded request is received, the receiver calls *best_neighbor* algorithm with sender as argument, so as to get a different neighbor. If a node is the originator of a request, it calls *best_neighbor* with itself as the argument. The algorithm attempts to find a neighbor other than the sender of the request. If no such neighbor can be found, the message is returned back to the sender. This happens only in case of nodes that have only one neighbor.

The parameters are *prev_nbr*: the neighbor to which request was earlier forwarded, *nbr*- a neighbor in neighbor list, *nbrcnt*- the number of neighbors for the node, *nbr[resp_time]*-average response time taken by *nbr*, *nbr[hits]*-the number of hits produced by *nbr*.

```
best_neighbor(prev_nbr)
{
if (experience-cache is empty && nbrcnt!=0)
select a random neighbor nbr;
```

```
best_nbr= nbr;

else if (nbrcnt>=2)
{
maxhits= hits of first nbr in the neighbor list;
best_response=MAX_RESPONSE_TIME;
best_nbr=first nbr in the neighbor list;

if (prev_nbr==me)
{
for each neighbor nbr in neighbor list
{
if (maxhits<nbr[hits])
{
maxhits=nbr[hits];
best_response=nbr[resp_time];
best_nbr=nbr;
}

else if (maxhits == nbr[hits])
{
if (nbr[resp_time]<best_response)
{
best_response=nbr[resp_time];
best_nbr=nbr;
}
else if (nbr[resp_time]==best_response)
randomly select one node;
}
}
}
else
{
for each neighbor nbr in neighbor list
{
if (nbr!=prev_nbr)
if (maxhits<nbr[hits])
{
maxhits=nbr[hits];
best_response=nbr[resp_time];
best_nbr=nbr;
}

else if (maxhits == nbr[hits])
{
if (nbr[resp_time]<best_response)
{
best_response=nbr[resp_time];
best_nbr=nbr;
}
else if (nbr[resp_time]==best_response)
randomly select one node;
}
}
}
}
```

```

}
return best_nbr
}

```

Further each matchmaker uses a **request-cache** that maintains details of forwarded requests. It contains *request id* -request identifier to keep track of requests, *forwarded neighbor*- the node to which request is forwarded, and *forward count* -number of times the request is freshly forwarded from the source. The parameter *forward count* has an upper limit that can be decided by the local matchmaker. We assume *forward count* is a locally configurable parameter. This in no way affects the resource discovery performance of other matchmakers. Finally, a *timestamp*-the time at which the request was forwarded for each forwarded request.

3.2. Request Forwarding Algorithm

The request forwarding algorithm employed in the framework is described below. Each matchmaker listens for incoming match requests. The matchmaker attempts to find a resource locally, by searching for a match in its local pool of advertisements. If a suitable match is not found, the request is forwarded to a remote matchmaker. If a match is found, the response is sent to the node from which request was originated. A request is discarded if its time to live (TTL) value expires. A *req_discard* message along with the details of the discarded request is sent to the request originator indicating the discard of its request. When a discard message is received by a match maker it reconstructs the same request and sends to a different neighbor. If the node has only one neighbor, it follows iterative deepening strategy, *i.e* increment the TTL value for each new forward and send to the same neighbor.

The algorithm is presented below

```

{
    M=receiveMessage();
    source=M.source;
    ttl=M.ttl;
    set MsgType type as M.type
    if (MsgType==req_discard or request timed out)
    {
        re construct the discarded request .
        if (nbrcnt==1)
        {
            //iterative deepening
            newTTL=oldTTL+1;
            if (newTTL<MAX_TTL_VALUE)
            Send to the same neighbor with newTTL;
            Update experience-cache and request- cache;

```

```

else
    giveup!!
}
else
{
    if (reqid[fwdcnt]< MAX_FORWARDS)
    forward to best_neighbor(M);
    update experience-cache;
    update request- cache;
    else
    giveup!!
}
}

else if (MsgType==Match Results)
{
    Add "source" to the neighbor list;
    Update experience-cache;
    Delete request from request- cache;
    Return match results to local requester;
}
else if (MsgType==Forwarded Request)
{
    Check for a match in local pool of ads;
    if (match)
    Send match results;
    else
    {
        ttl--;
        if(ttl>0)
        K=best_neighbor(M);
        Send request to K with Ttl;
    }
    else
    Send req_discard to source;
}
}
}

```

The values of MAX_TTL_VALUE and MAX_FORWARDS are decided by the network. These parameters are configurable depending on the response time and the discovery performance of the algorithm. The parameters affect the discovery performance of the individual matchmaker but do not affect the discovery performance of other matchmakers. MAX_TTL_VALUE parameter is utilized when the matchmaker has only one neighbor. Once the number of neighbors exceeds one, MAX_FORWARDS parameter is used. The average response time is updated each time the result is returned from a neighbor.

The algorithm ensures that every time a different neighbor is tried, when a request is once discarded. For every discarded message of a node, the node

maintains a list of forwarded neighbors (for nodes with more than one neighbors) to ensure that a different neighbor is tried every time. Another alternative to handle discarded messages could be selective flooding. We believe that this strategy is suitable only when the neighbor list updates are infrequent or rare. But in a grid environment neighbor list updates are common. So, a best neighbor can provide better information, than a flooded message that passes through only neighbors, but not directly to the provider.

Further the size of the experience-cache and request cache affects the performance of the resource discovery. The maximum size of experience-cache can be the number of matchmakers in the grid. This occurs when all other nodes in the grid know the node in context. But such an arrangement leads to sudden destabilization of the resource discovery system when the node fails. So, we suggest an upper bound on the maximum number of neighbors per each node. This bound directly influences the size of experience-cache. The size of request cache is dependent on the number of requests matchmaker generates in unit time. We believe that request forwarding is a service that should be provided by every member node (matchmaker) in the grid.

4. Experimental Evaluation

4.1. Experimental Setup

We evaluated our frame work on a cluster of work stations, each running matchmakers that perform ClassAds matchmaking. A starting topology for the network was generated with *ngce* [10], a tool to generate network graphs. The request forwarding algorithm is evaluated with 200 matchmakers. We made the following assumptions during the experiment. A hundred different classes of resources are uniformly distributed across the nodes with an average replication factor of 5. Average number of resources provided by a matchmaker is 5. We have considered fixed number of attributes for resource matching. Resource requests are generated based on the Zipf distribution. Hop count is used as a measure for TTL.

The request forwarding algorithm is evaluated in two phases, unlearned and learned respectively. In the unlearned phase, a selected number of nodes generate fixed number requests each. The resources are discovered based on random request forwarding and the nodes keep learning from the responses. During this phase nodes do not use experience-cache information to make forwarding decisions. In the learned phase, the same sets of requests are generated

to the same nodes utilizing the proposed request forwarding algorithm. We observe the response times and average TTL values and the network load come down as the nodes perform resource discovery using our algorithm. The graphs below indicate the results of our experiments.

4.2. Performance Analysis

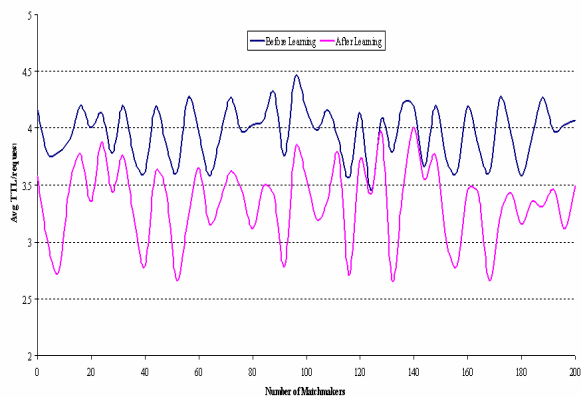


Figure 2. Average TTL per request versus the number of matchmakers

Figure 2 shows the average TTL per request in the unlearned and learned phases. In the unlearned phase, average TTL values are evenly distributed across the nodes (close to value 4 in figure). In the learned phase, we see the average TTL values have come down drastically for few nodes. These are the nodes that learned during unlearned phase. The more the nodes learn, lesser is the average TTL value per request.

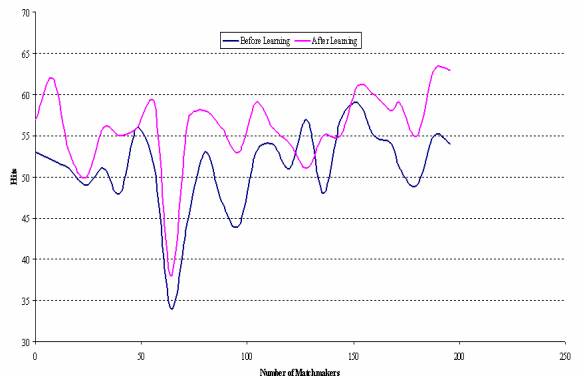


Figure 3. Number of hits versus number of matchmakers

Figure 3 depicts the comparison of number of hits during both the phases. Both the curves are almost similar, but in learned phase an upward shift indicates the increase in number of hits. Figure 4 indicates the number of forwards in both the phases. In unlearned

phase, all the nodes evenly share the forwarding load. In the other phase, few peaks are observed. These indicate that few nodes performed most of the forwarding. These nodes lie on the unique path to some popular (frequently asked) nodes. All other nodes do very minimum forwarding compared to the previous phase. On average we observe that the number of forwards is reduced significantly indicating the less network load imposed by the algorithm. Figure 5 depicts the average response times for different TTL values ranging from 1 to 5. The response times were observed in the learned phase.

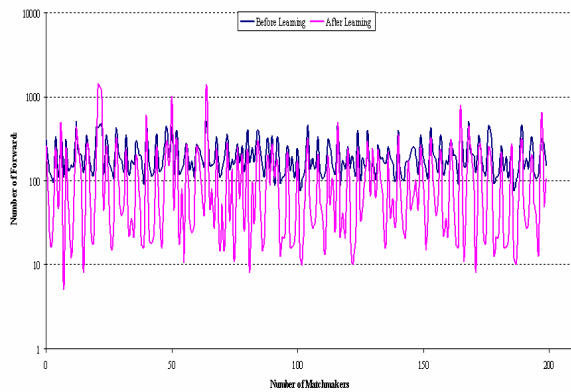


Figure 4. Number of forwards versus number of matchmakers

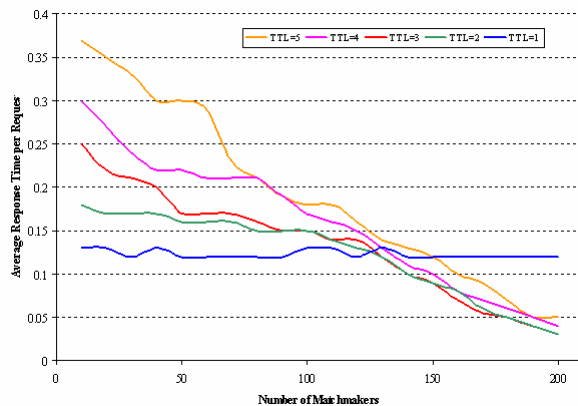


Figure 5. Average response time per request versus number of matchmakers for TTL =1 to 5

6. Conclusions and Future work

Discovering resources in a grid environment poses potential problems due to dynamic, scattered, distributively owned nature of resources. Moreover, providers and requesters of resources join and leave the grid arbitrarily. Current grid systems employ

centralized resource discovery mechanisms. Centralized resource discovery systems do not scale well with increasing number of participants of in the grid. The proposed approach attempts to take the advantage of matchmaking framework to implement a distributed resource discovery service and thus distributing the central matchmaker's load to withstand the overhead due to growing participants of the grid.

The proposed framework works well in a fault free grid system. Also, when a new node joins the grid, it is possible that our resource discovery mechanism may not discover its resources immediately as the number of hits against that node will be zero in all of its neighbors. Further, we are investigating mechanisms to include proximity matching in resource discovery service when suitable match information is not found. In such a case, the requester is negotiated and is asked to relax few constraints so as to obtain a better match. We would further evaluate the framework against various request and resource distributions. These issues will be addressed in our future work.

7. References

- [1] A. Iamnitchi and I. Foster. On Fully Decentralized Resource Discovery in Grid environments. *International Workshop on Grid Computing*, 2001.
- [2] C.Mastroianni, D.Talia, O.verta. A super peer model for building resource discovery services in grids-design and simulation analysis. *European Grid Conference 2005*.
- [3] I. Foster and C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufman, 2004.
- [4] The Condor homepage. <http://www.cs.wisc.edu/condor>
- [5] The ClassAds Homepage. <http://www.cs.wisc.edu/condor/classad>
- [6] Globus Toolkit website. <http://www.globus.org>
- [7] Gnutella protocol specification. <http://www.clip2.com/articles.html>.
- [8] Napster website. <http://www.napster.com>
- [9] European DataGrid project. <http://www.eu-datagrid.org>
- [10] Network graph for computer epidemiologists <http://www.dmst.aueb.gr/dds/pubs/conf/2005-PCI-NGCE/html/NGFF.html>
- [11] H.Zhughe and J.Liu, Flexible Retrieval of Web Services, *Journal of Systems and Software*, 70 (1-2) 2004, 107-116.