Grid service discovery
An approach using Parameter and QoS based matchmaking

S. Thamarai Selvi, R.A. Balachandar, Vyas Swaminathan, Ashwin Kumar, Venkatesh Paramasivan
Department of Information Technology
Madras Institute of Technology, Anna University
Chennai, India

Abstract—This paper addresses the problem of discovering grid services in a heterogeneous grid environment. Our system uses semantic descriptions of grid services using the WSDL-S tool. The matchmaking algorithm used in our system uses parameter based matchmaking with inputs, outputs and functionalities to match service advertisements and requests. In case of multiple service hits for a particular service advertisement, QoS based matchmaking is used to select a service.

Keywords—Matchmaking, Service Discovery, QoS

I. INTRODUCTION
A Grid instantiation is a grid system prototype using one or more grid middleware technologies. Most current grid instantiations are focused on computational services for end users [2]. Currently, Grid Services use metadata to describe and discover services. However, simple metadata and queries give only a small and insignificant improvement to information integration [3]. The Globus Toolkit’s Monitoring and Discovery System (MDS) defines and implements mechanisms for service discovery and monitoring in distributed environments [5, 8]. MDS uses symmetric and attribute based matchmaking [3, 20], where the values of attributes advertised by the nodes are compared with those required by jobs. However, for the comparison to be meaningful and effective in MDS, the node providers and consumers have to agree upon attribute names and values, i.e. the matchmaking system needs to be standardized across users and servers. It as cannot differentiate between different character sets or user representation formats. Hence, there is a demand for a framework enabling description and discovery of services in grid environments where knowledge of the capabilities of services can be used in discovery. Fundamental research on the semantic web has allowed the grid community to move from the current data centric view supporting the Grid, towards a Semantic Grid with a set of domain specific problem solving services and knowledge services [4, 5, 6]. The Semantic Grid supports flexible collaboration and computation on a global scale with a high degree of automation. In such an environment, it is essential to provide facilities to the user for easy discovery of the services and also to the service providers, a better way to describe their services. The matchmaking system used in this paper uses WSDL-S to add semantics to the grid service concepts. A parameter matchmaking algorithm is proposed which matches inputs, outputs and functionalities. This improves on traditional matchmaking techniques by allowing for semantic discovery of services. We have described a matchmaking algorithm which uses weighted QoS factors to break service discovery ambiguities [1]. In this paper, an integration of QoS based matching with parameter based matchmaking algorithm is demonstrated in order to improve the accuracy of retrieving a service which matches the user’s requirements. In addition, the weights in the QoS based matchmaking algorithm are no longer limited by the matchmaking system and the priorities are adjusted based on the values entered by the user. User-defined weighted QoS factors are used to obtain suitable services that meet the user’s requirements. QoS based matchmaking is performed only when the user’s requirements are satisfied to an equal extent by multiple services. The experimental results of this integration of the parameter matchmaking algorithm with the QoS based matchmaking algorithm have shown that there is a substantial improvement in the accuracy of finding a service which matches the users’ requirements when the QoS based matchmaking is used in addition to parameter matchmaking.

The organization of the paper is as follows: In section II, the related research work going on in the field of semantic grids is mentioned. In section III and IV, the architecture of the matchmaking system is discussed. Section V explains the matchmaking algorithm and section VI shows the experimental results that are obtained.

II. RELATED WORK
Literature [9, 10] implements a grid service Matchmaking system that works purely based on attributes. This is similar to the matchmaking tools available in MDS.

Services need to be described semantically for successful capability matching. Currently, WSDL specification [7] is used to describe service capabilities. However, it lacks the ability to provide semantic meaning to service descriptions. Literature [11] uses WSDL-S to enrich the semantics of the WSDL descriptions. The OWL-S semantic description language [12, 13] aims to increase the meaning of web services by annotating the data with Ontologies. However, OWL-S is more suited for the semantic annotations of web services than grid services. In literature [3], QoS
parameters are used for resource discovery. It proposes various parameters i.e. CPU cycles, bandwidth and memory space. In this paper, we consider the QoS parameters as suggested by [3]. QoS factors are generally defined for network based applications & there is no specific definition of these parameters for grid services. Our approach of integrating the weight based QoS parameters with grid service discovery improves the efficiency of the service discovery and involves the end user participation.

In addition to the parameters mentioned in literature [3], we consider network latency and reliability. The matchmaking algorithm proposed in this paper, matches the QoS factors if multiple advertised services meet the requested capabilities. It allows the service requester to associate a weight to the QoS factors defined by the matchmaking system.

III. MATCHMAKING

The discovery module matches the service requests against the service advertisements and returns the service that meets the requested capabilities. The matchmaking system compares the service request(R) given by the user with the service advertisements(S). The capability of a service can be expressed in terms of functionality that the service offers, the kind of input it takes and output it produces. Therefore, matchmaking is done based on functionalities, outputs and inputs (FOI), together known as parameters. With domain and function ontology in the background, it is possible to determine the semantic similarity between the parameters and determine the degree of match. Depending on the extent of the match, a service is classified into one of the following four kinds [15]:

1) Exact Match: This occurs if the advertised FOI of the service S exactly matches with that of requested service R.
   \[ R(FOI) = S(FOI) \rightarrow \{ R(F) = S(F) \ U R(O) = S(O) \ U R(I) = S(I) \} \]

2) Plug-in Match: This match occurs if S contains more FOI than that R requires. This match is obtained after referring the background ontology. If the requested concept is modeled as sub concept of the advertised concept in the respective background ontology, then there exists a plug-in match between the advertised and requested capabilities.
   \[ R(FOI) \leq S(FOI) \rightarrow \{ R(F) \leq S(F) \ U R(O) \leq S(O) \ U R(I) \leq S(I) \} \]

3) Subsume Match: This match occurs if the requested service R contains more FOI than the service advertisement S. In this case the service can only partially satisfy the user’s requirements.
   \[ R(FOI) \geq S(FOI) \rightarrow \{ R(F) \geq S(F) \ \& R(O) \geq S(O) \ \& R(I) \geq S(I) \} \]

4) Disjoint: The requested service R does not match with the described service S
   \[ R(FOI) \neq S(FOI) \rightarrow \{ R(F) \neq S(F) \ U R(O) \neq S(O) \ U R(I) \neq S(I) \} \]

Parameter matchmaking works well when considering the fulfillment of only the user’s service functionality requirements. We are faced with a problem when there are multiple services available to the user which offer the same or similar functionality. In this case, the user needs a way to select the best service among the many services available to him. Here, our QoS based matchmaking algorithm helps the user select the best service to satisfy his requirements. Different users may have different QoS requirements with respect to the service requested. A lot of research is going on in trying to solve this problem. Devising a generic QoS based matchmaking algorithm is a difficult task as users have varying QoS requirements. Our algorithm solves this problem by using several QoS factors associated with user-defined weights for matchmaking [14]. The discovery module allows the users to specify their own priorities in terms of weights to the QoS factors that the matchmaking system supports. For example, a user might require a highly reliable service provider more than one that could process the request fastest. In this case, he would prioritize reliability by giving it a higher weight than the other factors. Based on the weighted QoS values, the discovery module computes the overall QoS and selects the most appropriate service.

IV. WEIGHTED QOS FACTORS

Weighted QoS factors are used to resolve ambiguities due to multiple service hits for a given user request [1]. We model the QoS needed by a service request in terms of six different factors, CPU cycles (CPU), network bandwidth (NBW), disk space (DS), memory buffers (MB), reliability (REL) and network latency (NL). These factors can be computed as shown:

\[
\begin{align*}
\text{CPU} &= 1-(R_{\text{requiredCPU}} / S_{\text{availableCPU}}) \\
\text{NBW} &= 1-(R_{\text{requiredNBW}} / S_{\text{availableNBW}}) \\
\text{DS} &= 1-(R_{\text{requiredDS}} / S_{\text{availableDS}}) \\
\text{MB} &= 1-(R_{\text{requiredMB}} / S_{\text{availableMB}}) \\
\text{REL} &= 1-(R_{\text{requiredREL}} / S_{\text{availableREL}}) \\
\text{NL} &= 1-(R_{\text{requiredNL}} / S_{\text{availableNL}})
\end{align*}
\]

Where \( R_{\text{requiredCPU}} \) is the minimum number of CPU cycles requested, \( S_{\text{availableCPU}} \) is the CPU cycles available at the service provider and so on. We define reliability as the ratio between the number of successful responses from the service provider (NoS) and the total number of service requests to that service provider (ToS).

\[
\text{Reliability} = \frac{\text{NoS}}{\text{ToS}}
\]

The parameters required by the service request such as \( R_{\text{requiredCPU}} \) are obtained from the service requester. The requester specifies the priorities of the QoS in terms of weights (w) to select a single suitable service when the requested capabilities are met by more than one service. The parameters representing available resources are obtained from the service provider using the Network Weather Service (NWS) [14]. The matchmaking system computes the overall QoS requested by the user as a function of weighted QoS.
factors as shown in equation (2).

\[
\text{OverallQoS} = \left(\frac{1}{\text{w}}\right) \sum (w_1 \times \text{CPU}, w_2 \times \text{NBW}, w_3 \times \text{DS}, w_4 \times \text{MB}, w_5 \times \text{REL}, w_6 \times \text{NL})
\] (2)

where \( w_i \) is the weight associated with every QoS factor and \( w=\sum w_i \)

The user is allowed to specify individual weights for QoS factors. By allowing this, the user is given the flexibility to choose the service that matches his QoS requirements the best.

V. MATCHMAKING ALGORITHM

Service Advertisement \( S \)
Service Request \( R \)
Parameters: Inputs \( I_k \), Outputs \( O_k \), Functionalities \( F_k \)

Parse \( S \) into \((S_{I_1}, S_{I_2}, \ldots), S_{(O_1, O_2, \ldots)}\) and \((S_{F_1}, S_{F_2}, \ldots)\)

Parse \( R \) into \((R_{I_1}, R_{I_2}, \ldots), R_{(O_1, O_2, \ldots)}\) and \((R_{F_1}, R_{F_2}, \ldots)\)

//Inputs, Outputs and Functionalities

For \( i \leftarrow 1 \) to \( n \)

For \( j \leftarrow 1 \) to \( m \)

If \( (R_{I_i} = S_{I_j}) \)

\[ \text{InputMatch} \leftarrow \text{InputMatch} + 1 \]

If \( (R_{O_i} = S_{O_j}) \)

\[ \text{OutputMatch} \leftarrow \text{OutputMatch} + 1 \]

If \( (R_{F_i} = S_{F_j}) \)

\[ \text{FunctionalityMatch} \leftarrow \text{FunctionalityMatch} + 1 \]

//We assume that there are no duplicate values in the
//input, output or functionality sets

Next \( j \)
Next \( i \)

\[ \text{MatchSum} \leftarrow \text{InputMatch} + \text{OutputMatch} + \text{FunctionalityMatch} \]

// \( \sum \) \( n \) Calculates the number of elements in a set

If \( (\sum_{n} (S_{I}) = \sum_{n} (R_{I}) = \text{InputMatch} \) \& \( \sum_{n} (S_{O}) = \sum_{n} (R_{O}) = \text{OutputMatch} \) \& \( \sum_{n} (S_{F}) = \sum_{n} (R_{F}) = \text{FunctionalityMatch} \)

Match \leftarrow “Exact”
MatchRating \leftarrow 2

Else if \( (\sum_{n} (S_{I}) > \sum_{n} (R_{I}) = \text{InputMatch} \) \& \( \sum_{n} (S_{O}) > \sum_{n} (R_{O}) = \text{OutputMatch} \) \& \( \sum_{n} (S_{F}) > \sum_{n} (R_{F}) = \text{FunctionalityMatch} \)

Match \leftarrow “Plug-in”
MatchRating \leftarrow 2

//Both Exact and Plug-in matches are given a MatchRating
//of 2 as the user’s requirements are completely satisfied

Else if (\( \text{MatchSum} > 0 \))

Match \leftarrow “Subsume”
MatchRating \leftarrow 1

//User’s requirements are partially satisfied

Else

Match \leftarrow “Disjoint”

MatchRating\leftarrow 0

//User’s requirements are not satisfied

End if

Repeat for every Service Advertisement \( S_n \)

If (MatchRating>0)

SelectService \( S_n \leftarrow \text{Max} \) (MatchRating \( (S_1, S_2, \ldots) \))

//We would select the service which is the best match for
//the request

If (SelectService is not unique)

OverallQoS \( (S_1, S_2, \ldots) \)

SelectService \( S_n \leftarrow \text{Max} \) (OverallQoS \( (S_1, S_2, \ldots) \))

End if
Else

No match exists
End if

VI. ILLUSTRATION

A. Experimental Results of Parameter matching

Two grid services \( S_1 \) and \( S_2 \) are run on different test machines. Each service provides addition, subtraction, multiplication and division functions.

<table>
<thead>
<tr>
<th>Service Request</th>
<th>Degree of Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Addition, Subtraction</td>
<td>Plug in</td>
</tr>
<tr>
<td>2 Addition, Subtraction, Multiplication and Division</td>
<td>Exact</td>
</tr>
<tr>
<td>3 Addition, Subtraction, Square root</td>
<td>Intersection</td>
</tr>
<tr>
<td>4 Square root, Logarithm</td>
<td>Disjoint</td>
</tr>
<tr>
<td>5 Addition, Subtraction, Multiplication, Division, Logarithm</td>
<td>Subsume</td>
</tr>
<tr>
<td>6 Multiplication, Addition, division</td>
<td>Plug in</td>
</tr>
<tr>
<td>7 Square root</td>
<td>Disjoint</td>
</tr>
<tr>
<td>8 Addition, Square root</td>
<td>Intersection</td>
</tr>
</tbody>
</table>

B. Experimental Results of the Weighted QoS comparison

A simulation of the effect of user defined weights on the two identical services \( S_1 \) and \( S_2 \) running on different servers presents the following results:

Let \( AS_n \) be the set of QoS factors of the service \( S_n \) as

\[ AS_n = \{CPU, NBW, DS, MB, REL, NL\} \]

The values of the sets \( AS_1 \) and \( AS_2 \) for \( S_1 \) and \( S_2 \) are:

\[ AS_1 = \{0.1, 0.3, 0.4, 0.5, 0.9, 0.8\} \]
\[ AS_2 = \{0.8, 0.1, 0.5, 0.4, 0.3, 0.1\} \]
\( w_i \): weights assigned to QoS factors

**Case (i)**

If the user does not specify any weights for the QoS factors that he requires, or if he specifies equal priorities for all the factors, then the OverallQoS is

<table>
<thead>
<tr>
<th>CPU</th>
<th>NBW</th>
<th>DS</th>
<th>MB</th>
<th>REL</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS1</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
<td>0.5</td>
<td>0.9</td>
</tr>
<tr>
<td>( w_i )</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>( AS_1 \ast w_i )</td>
<td>0.5</td>
<td>2.5</td>
<td>2.0</td>
<td>2.5</td>
<td>4.5</td>
</tr>
</tbody>
</table>

\[ \sum (AS_1 \ast w_i) = 15 \]

\[ w = \sum w_i = 25 \]

OverallQoS = \( (1/w) \ast \sum (AS_1 \ast w_i) \)

\[ = (1/25) \ast 15 \]

\[ = 0.6 \]

**Case (ii)**

Let us consider another case in which the user wishes to give CPU cycles and NBW a higher priority than the other four factors. With the same set of QoS factors for both \( S_1 \) and \( S_2 \), we compute overall QoS as shown below:

<table>
<thead>
<tr>
<th>CPU</th>
<th>NBW</th>
<th>DS</th>
<th>MB</th>
<th>REL</th>
<th>NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS2</td>
<td>0.8</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>( w_i )</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>( AS_2 \ast w_i )</td>
<td>4.0</td>
<td>0.5</td>
<td>2.5</td>
<td>2.0</td>
<td>1.5</td>
</tr>
</tbody>
</table>

\[ \sum (AS_2 \ast w_i) = 11 \]

\[ w = \sum w_i = 25 \]

OverallQoS = \( (1/w) \ast \sum (AS_2 \ast w_i) \)

\[ = (1/25) \ast 11 \]

\[ = 0.44 \]

With the OverallQoS obtained for \( S_1 \) and \( S_2 \), we observe that when the QoS factors are assigned equal weights, where the user does not prioritize his requirements (default condition) and the matchmaking selects \( S_1 \) as the best match for the user’s requirements.

**Comparison of Overall QoS of two Services when CPU weight is varied**

This graph shows the change in OverallQoS of the two services when the CPU weights are increased in steps of one from one to ten. The other QoS factors are given weights of 3, 3, 4, and 4 respectively.
D. Improvement in performance due to QoS based matchmaking

The ability of the service to match the users’ requirements was evaluated by attaching a feedback form to each service discovered for a user. The users could choose if they were satisfied or unsatisfied with the service. For this survey, 60 services, both identical and unidentical were distributed across 15 servers. The accuracy of retrieving a service is defined as a ratio between the number of satisfied responses and the total number of responses. The number of requests was twice the number of services.

![Figure 3: Comparison of Accuracy](image)

The results show that the QoS based matchmaking algorithm, when integrated with parameter matchmaking provides better results than if parameter matchmaking is used alone.

VII. CONCLUSION

In this paper, the integration of the QoS and parameter based matchmaking algorithm improves the efficiency of service discovery in heterogeneous grids. The QoS based matchmaking algorithm provides a flexible discovery mechanism and when used along with the parameter matchmaking algorithm helps in providing the best service to satisfy the users’ needs.

ACKNOWLEDGMENT

The authors express their sincere thanks to the Centre for Development of Advanced Computing (CDAC), India for providing financial assistance to this research project.

REFERENCES


