International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:1, 2007

Resource Matching and a Matchmaking Service for an Intelligent Grid

Xin Bai, Han Yu, Yongchang Ji, and Dan C. Marinescu

***Abstract*—**We discuss the application of matching in the area of resource discovery and resource allocation in grid computing. We present a formal definition of matchmaking, overview algorithms to evaluate different matchmaking expressions, and develop a matchmaking service for an intelligent grid environment.

***Keywords*—**Grid, Matchmaking, Ontology

I. Introduction

WEBSTER dictionary defines the verb "to match" as "to be equal, similar, suitable, or corresponding to in some way". In computer science, the term *matching* refers to a process of evaluation of the degree of similarity of two objects. Each object is characterized by a set of properties/attributes; each property is a tuple (*name*, *value*), with *name* a string of characters and *value* either a constant (a number – integer or real, a Boolean constant – "true" or "false", or a string of characters), or an expression that returns a constant. The "matching degree", *m*, is a real number. Typically, $0 \leq m \leq 1$, with $m=0$ meaning a *total mismatch* and $m=1$ a *perfect match*.

Matching is a common operation in many areas of computer science. In this paper we discuss the application of matching in the area of resource discovery and resource allocation in grid computing.

A *grid* is an open system, a large collection of autonomous systems giving individual users the image of a single virtual machine with a rich set of hardware and software resources. A set of *core grid services* provide access to various grid resources. For example, a *resource discovery* service assists users, or their proxies, to locate needed resources on the grid. In more traditional computing systems, resources are managed centrally under the control of a single administrative authority by the resource management component of an operating system or by a distributed operating system. The central management of resources on a grid is unthinkable because of the scale of the system and because it would violate the autonomy of individual resource providers, a critical aspect of grid.

The grid matchmaking process involves three types of agents: consumers (*requesters)*, producers (*providers)*, and a *matchmaking service*. A matchmaking service uses a *matching algorithm* to evaluate a *matching function* which computes the *matching degree*, see Fig.1.

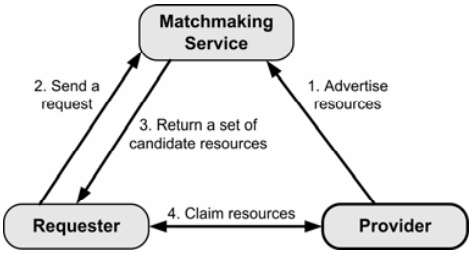

Fig. 1. The four-step grid matchmaking process: 1) Providers send resource descriptions to the matchmaking service; 2) A request is sent to the matchmaking service; 3) The matchmaking service executes a matchmaking algorithm and returns a set of ranked resources to the requester; 4) The requester chooses a resource from the set the contacts the corresponding resource provider.

In this paper we introduce several types of resource matching functions for a grid environment and discuss briefly a matchmaking service. In a related paper [1] we present an evaluation of this service.

II. Related Work

MATCHMAKING has been a hot topic of *MAS* (*Multi-Agent Systems*) research, related to question on how to find a suitable agent for a specific problem. The notable results in this area are *ACLs* (A*gent Communication Languages*) and matchmaking algorithms based on these languages. One of the earliest results in this area is *ABSI* [2] (*Agent-Based Software Interoperability*) facilitator which uses *KQML* (*Knowledge Query and Manipulation Language*) specification and *KIF* (*Knowledge Interchange Format*) as the *content language*. The matchmaking of an advertisement and a request is made through the unification of equality predicate. In the *COIN* system [3], the matchmaking algorithm is based on a unification process similar to *Prolog*. The *InfoSleuth* [4] uses *KIF* as the content language. The matchmaking algorithm is based on constraints, i.e., the advertisement and the request match if the constraints are satisfied. *SDL* (*Service Description Language*) was proposed in [5] to describe

Manuscript received December 16, 2004. This work was supported in part by National Science Foundation grants MCB9527131, DBI0296107, ACI0296035, and EIA0296179.

Xin Bai, Han Yu, and Yongchang Ji are with the School of Computer Science, University of Central Florida, P. O. Box 162362, Orlando, FL 32816-2362. (e-mail: [xbai, hyu, yji]@cs.ucf.edu).

Dan C. Marinescu is a professor in the School of Computer Science, University of Central Florida, P. O. Box 162362, Orlando, FL 32816-2362. (phone: 407-823-4860; fax: 407-823-5419; e-mail: dcm@cs.ucf.edu).

services. The matchmaking algorithm finds *k*-nearest services for a request according to the distance between the service names (pairs of verb and noun terms) and the request. *CDL* (*Capability Description Language*) was proposed in [6]. It supports reasoning through the notions of *subsumption* and *instantiation*. *LARKS* (*Language for Advertisement and Request for Knowledge Sharing*) was proposed in [7] for describing service capability and service request. It supports *ITL* (*Information Terminological Language*) [8] concept language. The relations among concepts are used to compute semantic similarities. The matchmaking in *MAS* involves semantic service matchmaking using the concept relationship and word distance to determine the semantic similarities of advertisements and requests. The matchmaking in *MAS* does not involve other resource types and the matchmaking results are exact, i.e., only "true" and "false" are allowed.

Research for service discovery in the Internet involves ontology-based matchmaking. The traditional methods of service discovery include name matchmaking and keyword matchmaking. Some new methods are based on ontologies. In [9] a semantic matchmaking framework based on *DAML-S*, a *DAML* (*DARPA Agent Markup Language*)-based language for service description, was proposed for semantic matchmaking of web services capabilities. The basic idea is that an advertisement matches a request when the service provided by the advertisement can be of some use to the requester. The matchmaking is performed on the outputs and inputs of the advertisement and the request based on the ontologies available to the matchmaker. Through the subsumption relationship of one concept of the input/output of the advertisement and one concept of the input/output of the request, four levels of matching can be determined: exact, plug-in, subsume, and fail. The idea of checking the concepts of input and output is similar to the one in the MAS research.

The matchmaking framework of *Condor* [10] system uses a semi-structured data model [11] called classified advertisements (classads) to describe resources and requests. A *classad* is a mapping from attribute names to expressions. *Condor* matchmaking takes two classads and evaluates each one with the other. A classad has an attribute named "Constraint" that is used to be evaluated in the context of this classad and the classad being matched with this classad. Only when the values of attribute "Constraint" of both classads are evaluated to be true, can these two classads be thought to be matched. A classad has an attribute named "Rank" that measures the desirability of a match. The evaluation value of "Rank" identifies how much the two classads match. The larger the value, the better they match. The *Condor* system requires the provider and the requester to know each other's classad structure. The evaluation result of the attribute "Rank" is generally not normalized and can not tell explicitly how well two classads match.

The matchmaking framework in *Condor* supports the selection of only one resource. Based on the *Condor* matchmaking, in [12], an extension, called set-extended classad syntax, was proposed to support the multiple resource selection. The matchmaking algorithm evaluates a set-extended *classad* with a set of *classads* and returns a *classad* set with the highest rank. When the size of the *classad* set is large, it is not feasible to evaluate all of the possible combinations of the resources. A greedy heuristic algorithm is used to find the *classad* set with the highest rank. This set-extended resource selection framework can perform both resource discovery and resource allocation.

In [13], a service selection model was proposed based on the quality of service of different service providers. Each time after a service is used the user sends a feedback to the matchmaker. When a request arrives, the matchmaker finds the most appropriate service according to the inputs specified by the request based on the feedbacks it receives.

III. The Grid Matchmaking Problem

Given a request and a set of resources, our goal is to find a set of resources that best match the request. A *request* is a (n+1)-tuple consisting of n attributes (a_1, a_2, …, a_n) and a function of these attributes to be evaluated in the context of resources, i.e., request = [a_1, a_2, ..., a_n, f(a_1, a_2, ..., a_n)]. An attribute of a request is a mapping from an attribute name to an attribute expression. A *resource* is an m-tuple consisting of m attributes (a_1, a_2, ..., a_m). An attribute of a resource is a mapping from an attribute name to an attribute value. The resource that returns the largest value of function f is considered as the one that best matches the request.

Attribute names should be constructed according to the corresponding resource ontologies. Resource ontologies are a critical component of the matchmaking framework. An *ontology* is an explicit specification of a conceptualization and a conceptualization is an abstract and simplified view of the world that we wish to represent for some purpose. An ontology defines a common structure that facilitates the sharing of information. It includes machine-interpretable definition of the basic concepts in a domain and their relations. Entities in the matchmaking framework, i.e., the providers, the requesters, and the matchmaking services, which are generally not in the same domain, must share the same ontology structure.

The structure of a category of entities is described as a Protégé *class* [14]. Fig. 2 shows the hierarchical relationship among grid resource classes. A class consists of one or more slots. A *slot* describes one attribute of the class and consists of a name and a value. An instantiation of a class is called an *instance* of that class. The type of a slot value may be simple types, such as integer, float, Boolean, and string, or an instance of a class.

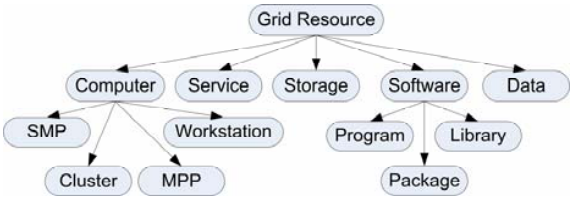

Fig. 2. The hierarchical relationship among grid resource classes.

The rules for constructing the attribute name for an attribute are:

1. If the attribute refers to a slot of the resource class, the attribute name is the slot name.
2. If a slot of the resource class refers to an instance of other class and the attribute refers to a slot of this instance, the attribute name is the combination of the two slot names connected by ".".

The attribute names of a cluster in Fig. 3 are constructed according to the above rule.

```
Boticelli{                          Bond{
  ResourceType=Cluster                ResourceType=Cluster
  ResourceName=Boticelli              ResourceName=Bond
  IPAddress=botticelli.cs.ucf.edu     IPAddress=bond.cs.ucf.edu
  NodeMemory.Size=3 GB                NodeMemory.Size=4 GB
  NodeCPU.Speed=1.6 GHz               NodeCPU.Speed=3.6 GHz
  NodeHarddisk.Capacity=80 GB         NodeHarddisk.Capacity=120 GB
  NumberOfNodes=44                    NumberOfNodes=32
  TotalMemorySize=132 GB              TotalMemorySize=128 GB
  TotalCPURates=140.8 GHz             TotalCPURates=115.2 GHz
  TotalDiskCapacity=3520 GB           TotalDiskCapacity=3840 GB
  ...                                 ...
}                                   }
```

Fig. 3. Example of a resource instance: the cluster *Boticelli* and *Bond.*

For a request = [a_1, a_2, ..., a_n, $f(a_1, a_2, ..., a_n)$], the function f is an expression that is the combination of attribute expressions $f_1(a_1)$, $f_2(a_2)$, ..., and $f_n(a_n)$ through mathematical and/or logical operators, where $f_1(a_1)$, $f_2(a_2)$, ..., and $f_n(a_n)$ are to be evaluated in the context of the corresponding attributes of the resource.

The expression for function f may involve:

1. Boolean expressions can be combined with the use of Boolean operators "&" and/or "|". Fig. 4(a) is an example for this case.
2. Arithmetic expressions can be combined with the use of arithmetic operators, such as "+", "-", "*", and "/". Fig. 4(b) is an example for this case.
3. Fuzzy expressions can be combined with the use of fuzzy operators "&&". The evaluation result of multiple fuzzy numbers connected by "&&" are the average of these fuzzy numbers. Fig. 4(c) is an example of this case.
4. A Boolean expression can be combined with an arithmetic expression or a fuzzy expression through Boolean operator "&". If the Boolean expression returns 1, they are evaluated to the value returned by the arithmetic expression or the fuzzy expression. If the Boolean expression returns 0, they are evaluated as 0. Fig. 4(b) and Fig. 4(c) are examples for this case.
5. Expressions are combined through "*if*", "*then*", and "*else*" constructs. Fig. 4(b) and Fig. 4(c) are examples for this case.

We define three types of matching functions. A matching function f may contain Boolean expressions and return a Boolean constant ("true", 1 or "false", 0), see Fig. 4(a). f may also contain arithmetic expressions and return a positive real number, see Fig. 4(b). We also allow f to contain fuzzy expressions and return a fuzzy number in [0, 1], as shown in Fig. 4(c). The higher the returned value, the better a request can be satisfied.

```
Request1 {
 f = ( ResourceType == "Cluster" ) &
     ( NodeCPU.Speed >= 1.6 GHz ) &
     ( if ( NodeMemory.Size < 2 GB ) then ( NumberOfNodes > 30 )        (a)
       elseif ( NodeMemory.Size < 3 GB ) then ( NumberOfNodes > 20 )
       else ( NumberOfNodes > 10 ) )
}
Request2 {
 f =  ( ResourceType == "Cluster" ) &
      ( ( TotalMemorySize/40 GB ) + ( TotalCPURates/50 GHz ) +          (b)
        ( TotalDiskCapacity/30 GB ) )
}
Request3 {
 f = ( ResourceType == "Cluster" ) &
     ( NodeMemory.Size >=1 GB ) &
     ( NodeCPU.Speed >= 1.6 GHz ) &
     ( NodeHarddisk.Capacity >= 30 GB ) &                               (c)
     ( TotalDiskCapacity > 40 GB ) &
     (
         ( if ( NodeMemory.Size > 4 GB ) then 1
           else ( NodeMemory.Size / 4 GB ) ) &&
         ( if ( NumberOfNodes > 40 ) then 1
           else ( NumberOfNodes / 40 ) )
     )
}
```

Fig. 4. Boolean, arithmetic, and fuzzy requests.

To evaluate f of Fig. 4(a) in the context of the two clusters in Fig. 3, *Boticelli* and *Bond* returns 1. To evaluate f of Fig. 4(b) in the context of *Boticelli*, "f = 1 & (132 / 40) + (140.8 / 50) + (3520 / 30) = 123.45". To evaluate f of Fig. 4(b) in the context of *Bond*, "f = 1 & (128 / 40) + (115.2 / 50) + (3480 / 30) = 121.5". The cluster that best matches the request is *Boticelli*. To evaluate f of Fig. 4(c) in the context of *Boticelli*, "f = 1 & 1 & 1 & 1 & (3 / 4) && 1 = 0.875". To evaluate f of Fig. 4(c) in the context of *Bond*, "f = 1 & 1 & 1 & 1 & 1 && 1 = 1". The cluster best matching the request is *Bond.*

IV. A MATCHMAKING SERVICE

We implemented a matchmaking service in an intelligent grid environment, the BondGrid [15]. The matchmaking framework includes a resource specification component, a request specification component, and matchmaking algorithms. A request specification includes a matchmaking function and possibly two additional constraints, a *cardinality threshold* and a *matching degree threshold*. The cardinality threshold specifies how many resources are expected to be returned by the matchmaking service. The matching degree threshold specifies the least matching degree of one of resources returned by the service.

The matchmaking service executes a matchmaking algorithm for each request sent by the requester. The input of the algorithm is the request and the grid resource instances stored in the knowledge base of the matchmaking service. The matchmaking algorithm evaluates the request function in the context of each resource instance in the knowledge base. The output of the algorithm is a number of grid resources ranked according to their matching degrees. Let n denote the

cardinality threshold specified by the request. The matchmaking algorithm returns the grid resources that have the *n* largest matching degrees to the requester. Fig. 5 shows the pseudo code of the matchmaking algorithm.

```
MATCHMAKING ALGORITHM
INPUT request req, a finite set of resource instances rs
OUTPUT a finite set of candidate resource instances cs
BEGIN
   cs = Φ
   n = req.CardinalityThreshold
   m = req.MatchingDegreeThreshold
   FOR each resource r in rs
      md = evaluate req.RequestFunction in the context of r
      IF (md>0) AND (md>=m)
         add r into cs
      END IF
   END FOR
   sort items in cs according to their matching degrees
   keep the items in cs that have the highest n matching degrees and remove the rest
END
```

Fig. 5. The original matchmaking algorithm performs an exhaustive database search.

A matchmaking service maintains a knowledge base with a large number of resource instances. Performing an exhaustive matchmaking involving all resources in the knowledge base is very expensive for large knowledge bases. In a modified matchmaking algorithm shown in Fig. 6, the algorithm finishes searching the knowledge base when $k*n$ (where k is a constant) resources are found with the required matching degrees (not less than the matching degree threshold).

```
MATCHMAKING ALGORITHM
INPUT request req, a finite set of resource instances rs
OUTPUT a finite set of candidate resource instances cs
BEGIN
   cs = Φ
   n = req.CardinalityThreshold
   m = req.MatchingDegreeThreshold
   FOR each resource r in rs from a random beginning position
      md = evaluate req.RequestFunction in the context of r
      IF (md>0) AND (md>=m)
         add r into cs
      ENDIF
      IF ( the size of the candidate set > k * n )
         break
      ENDIF
   END FOR
   sort items in cs according to their matching degrees
   keep the items in cs that have the highest n matching degrees and remove the rest
END
```

Fig. 6. The modified matchmaking algorithm performs a restricted database search; it stops when the cardinality of a set of resources that match the request reaches $k*n$.

We conducted performance studies regarding the variation of the response time with the size of the knowledge base in BondGrid environment [15]. Our measurements indicate that when the knowledge base contains more than 3000 resource records it is preferable to use a database rather than a local file. The measurements also show that requests with different levels of complexity of the evaluation function have similar response time. This indicates to us that the complexity of a request has little effect on the efficiency of the algorithm; indeed, the knowledge base access time is the major contributor to the response time. The response time of a matchmaking service running the modified matchmaking algorithm in Fig. 6, is considerably lower than that of a service running the original matchmaking algorithm in Fig. 5. The modified algorithm greatly improves the efficiency of matchmaking without sacrificing the quality of the matchmaking results.

V. SUMMARY

In this paper we introduce for the first time a more comprehensive resource matching scheme that supports a variety of matching functions including fuzzy functions. We implemented a matchmaking service for an intelligent grid environment and our evaluation results indicate that the complexity of the matching function has little effect on the response time dominated by the network and database access time.

REFERENCES

[1] X. Bai, H. Yu, Y. Ji, and D. C. Marinescu. “Evaluation of a matchmaking service for a grid environment.” The Fourteenth International Heterogeneous Computing Workshop, Denver, Colorado, April 4, 2005(submitted).

[2] N. Singh. "A common Lisp API and facilitator for ABSI: version 2.0.3." Technical Report Logic-93-4, Logic Group, Computer Science Department, Stanford University, 1993.

[3] D. Kuokka and L. Harada. “Matchmaking for information agents.” Proceedings of 14th IJCAI, 672-678, 1995.

[4] R. J. Bayardo Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. “InfoSleuth: agent-based semantic integration of information in open and dynamic environments.” Reading in Agents, Morgan Kaufmann, CA, 205-216, 1998.

[5] V. S. Subrahmanian, J. Dix, and F. Ozcan. “Heterogeneous agent systems.” The MIT Press, 2000.

[6] G. J. Wickler. “Using expressive and flexible action representations to reason about capabilities for intelligent agent cooperation.” PhD thesis, University of Edinburgh, 1999.

[7] K. Sycara, S. Wido, M. Klusch, and J. Lu. “LARKS: dynamic matchmaking among heterogeneous software agents in cyberspace.” Autonomous Agents and Multi-Agent Systems, 5, 173-203, 2002.

[8] K. Sycara, J. Lu, and M. Klusch. “Interoperability among heterogeneous software agents on the Internet.” Carnegie Mellon University, PA, Technical Report CMU-RI-TR-98-22.

[9] M. Paolucci, N. Srinivasan, K. Sycara, and T. Nishimura. "Towards a semantic choreography of Web services: From WSDL to DAML-S." Proceedings of ICWS'03.

[10] R. Raman, M. Livny, and M. Solomon. “Matchmaking: distributed resource management for high throughput computing.” In Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 1998.

[11] S. Nestorov, A. Abiteboul, and R. Motwani. "Inferring structure in semistructured data." In Proceedings of the Workshop on Management of Semistructured Data, Rucson, Arizona, May 1997.

[12] C. Liu, L. Yang, I. Foster, and D. Angulo. “Design and evaluation of a resource selection framework for grid applications.” In Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing, 2002.

[13] Z. Zhang and C. Zhang. “An improvement to matchmaking algorithms for middle agent.” AAMAS, Bologna, Italy, 2002.

[14] http://protege.stanford.edu/

[15] X. Bai, H. Yu, G. Wang, Y. Ji, D. C. Marinescu, G. M. Marinescu, and L. Bölöni. “Intelligent grids.”, “Grid computing: software environments and tools”, Jose C. Cunha and O.F. Rana Eds, Springer Verlag, Heidelberg, 2004 (in print).