Resource Management through Multilateral Matchmaking

Rajesh Raman, Miron Livny and Marvin Solomon
University of Wisconsin, 1210 West Dayton Street, Madison WI 53703
{raman,miron,solomon}@cs.wisc.edu

Abstract

Federated distributed systems present new challenges to resource management, which cannot be met by conventional systems that employ relatively static resource models and centralized allocators. We previously argued that Matchmaking provides an elegant and robust resource management solution for these highly dynamic environments [5]. Although powerful and flexible, multiparty policies (e.g., co-allocation) cannot be accomodated by Matchmaking. In this paper we present Gang-Matching, a multilateral matchmaking formalism to address this deficiency.

1. Matchmaking

Resource management via Matchmaking occurs as a four-step process. Entities (i.e., servers and customers) requiring matchmaking services express their characteristics, constraints and preferences to a Matchmaker in *classified advertisements* (Step 1) . Figure 1 shows a workstation's classad, where the `Constraint` and `Rank` expressions identify the machine's constraints and preferences respectively. Attributes of candidate classads are accessed via the pseudo-attribute `other`. The Matchmaker employs a very generic *matchmaking algorithm* to create matches incorporating the constraints and preferences of entities (Step 2). Matched entities are then notified, and their classads discarded (Step 3). Finally, matched entities establish an allocation through a claiming process that does not involve the Matchmaker (Step 4).

Many complex and useful policies may be defined within this framework; interested readers are referred to reference [6] for sophisticated real-world examples.

```
[ Type      = "Machine";
  KeybrdIdle= '00:23:12'; // h:m:s
  Disk      = 323.4M;     // mbytes
  Memory    = 256M;       // mbytes
  LoadAvg   = 0.042969;
  Arch      = "INTEL";
  OpSys      "LINUX";
  KFlops    = 21893;
  Name      = "foo.cs.wisc.edu";
  Rank      = 1G - other.ImageSize;//smaller is better
  Constraint= other.Type=="Job" && other.Owner!="rival"
            && LoadAvg < 0.3 && KeybrdIdle>'00:15'
]
```

Figure 1. Classad describing a Machine

2. The License Management Problem

Many programs employ software licenses to implement program use policies. For example, the program may be valid only to certain users, or on certain workstations or subnets. Software licenses must therefore be managed as first-class resources. However, the Matchmaking solution to this problem requires *three* participants in the match (i.e., job, workstation and license), which cannot be accomodated by conventional (bilateral) Matchmaking.

We consider the following specific problem: A job requires a workstation and a software license to run successfully. However, there are a limited number of licenses, each of which is valid only on certain workstations.

The gang-matching solution to this problem may be easily generalized to enforce more sophisticated license management policies, and solve more general co-allocation problems, but a detailed discussion of these issues is beyond the scope of this paper.

3. Gang-Matching

The gang-matching extension replaces a regular classad's single implicit bilateral match imperative with an explicit *list* of required bilateral matches. The classad representing the gang-match request for the Job-Workstation-License example is illustrated in Figure 2.

The `Ports` attribute is a list that represents the matches required to satisfy the job. In the gang-matching model, bilateral matching occurs between ports of classads instead of entire classads themselves. Each port defines a `Label` which names the candidate bound to that port, replacing the fixed `other` pseudo-attribute. The scope of a label extends from the port of declaration to the end of the port list. Thus,

```
[ Type  = "Job";
  Owner = "raman";
  Cmd   = "run_sim";
  Ports = {
   [   // request a workstation
     Label = "cpu";
     ImageSize = 28M;
     Rank = cpu.KFlops/1E3 + cpu.Memory/32;
     Constraint =
       cpu.Type=="Machine" && cpu.Arch=="INTEL" &&
       cpu.OpSys=="LINUX" && cpu.Memory>=Imagesize;
   ],[   // request a license
       Label = "license";
       Host = cpu.Name; // cpu name
       Rank = 0;
       Constraint =
         license.Type=="License" && license.App==Cmd;
       ] } ]
```

Figure 2. A gang-match request

```
[ Type  = "License";
  App   = "sim_app";
  ValidHost= "foo.cs.wisc.edu";
  Ports = { [ Label = "requester";
              Rank  = 0;
              Constraint=requester.Type=="Job"
                  && requester.Host==ValidHost
             ] } ]
```

Figure 3. License Advertisement

expressions in the "license" port can refer to the "cpu" label but not *vice versa*. Port labels are private and local to the hosting classad to prevent namespace pollution and collisions.

Since label scopes extend beyond the port of declaration, information may be conveyed from one match locality to another. From Figures 2 and 3, we see that the license's constraint on `requester.Host` is conveyed to `cpu.Name` by the job classad. Thus, the given license is only valid on the machine "foo.cs.wisc.edu."

3.1. Creating Gang-Matches

Intuitively, a gang-match is obtained by binding each unbound port of a classad to a compatible port of a new classad until all ports are bound. Our algorithm proceeds by picking job classads in priority order and then using a top-down, backtracking algorithm to marshall the required gang. Since only a single job is served at a time, deadlock is prevented. Partial evaluation and indexing of both attributes and constraints facilitate efficient identification of compatible ports.

4. Related Work

Matchmaking is widely studied in agent systems. The advertising languages of ACL [2] and RETSINA [7] support reasoning so that very general behaviors may be described and inferred. In contrast to these knowledge-base representations, classads employ a database representation.

Many resource management systems [8, 4, 3] process jobs by using resources that are identified explicitly through a job control language, or implicitly, by submitting the job to a queue associated with a resource set. Jobs requiring multiple resources must be submitted to special queues — there is no general mechanism to marshal a unique mix of resources. In Globus [1], customers describe required resources in a resource specification language (RSL) based on a pre-defined schema of the resources database. However, resources cannot place constraints on requests, precluding policies such as the Job-License-Workstation example.

5. Conclusions and Future Research

Bilateral matchmaking is implemented and heavily used in the production releases of the Condor system. The flexibility and generality of this approach in highly dynamic and distributed environments have been experienced in practice. We are in the process of integrating gang-matching extensions to the bilateral framework. We are also investigating optimization techniques to increase the efficiency of identifying gangs, and defining improved protocols for advertising and claiming.

References

[1] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems.

[2] M. Genesereth, , N. Singh, and M. Syed. A distributed anonymous knowledge sharing approach to software interoperation. In *Proc. of the Int'l Symposium on Fifth Generation Computing Systems*, pages 125–139, 1994.

[3] R. Henderson and D. Tweten. Portable Batch System: External reference specification. Technical report, NASA, Ames Research Center, 1996.

[4] B. C. Neumann and S. Rao. The prospero resource manager: A scalable framework for processor allocation in distributed systems. *Concurrency: Practice and Experience*, June 1994.

[5] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high-throughput computing. In *Proc. of the 7th IEEE Int'l Symp on High Performance Distributed Computing (HPDC7)*, July 1998.

[6] R. Raman, M. Livny, and M. Solomon. Matchmaking: An extensible framework for distibuted resource management. *Cluster: Journal of Software, Networks and Applications*, 2(2), 1999.

[7] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, pages 36–46, dec 1996.

[8] S. Zhou. LSF: Load sharing in large-scale heterogenous distributed systems. In *Proc. Workshop on Cluster Computing*, 1992.