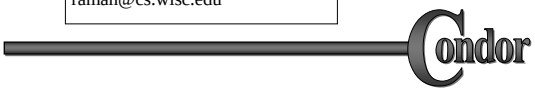
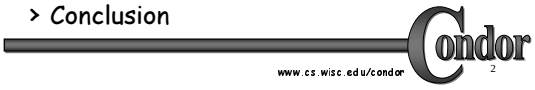
Matchmaking in the Condor System

Rajesh Raman
Computer Sciences Department
University of Wisconsin-Madison
raman@cs.wisc.edu



Outline

- Introduction to Matchmaking
  - Architecture, Philosophy
- Classified Advertisements (ClassAds)
  - Language for Matchmaking
- Matchmaking in Condor
- Future Directions in Matchmaking
- Conclusion



Matchmaking

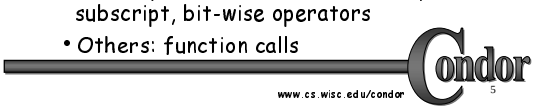
- Matchmaking is a methodology for Distributed Resource Management
- Conceptually simple:
  - Service providers and requesters advertise
  - Compatible advertisements are matched
  - Matched entities cooperate to perform service
- Developed for opportunistic environments
  - Use resources as and when available

Matchmaking (Cont.)

- Customers and Servers advertise to a Matchmaking Service
- Advertisements describe advertising entities
  - Characteristics
  - Requirements and Constraints
  - Preferences
- We call these descriptions classified advertisements (classads)

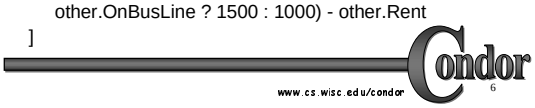
ClassAd Language Versions

- New ClassAd implementation will be released with Condor Version 6.5
- Older implementation lacks:
  - Some data types: lists, nested classads, time stamps, time intervals
  - Some operators: conditional, array subscript, bit-wise operators
  - Others: function calls



Example ClassAds

```
[
  Type = "Tenant";
  Name = "John Doe";
  Age = 28;
  HavePets = false;
  RentOffer = 900;
  Requirements = other.Type=="Apartment" &&
    other.Rent <= RentOffer && other.HeatIncluded;
  Rank = (other.Location=="Downtown" ? 2000 :
    other.OnBusLine ? 1500 : 1000) - other.Rent
]
```



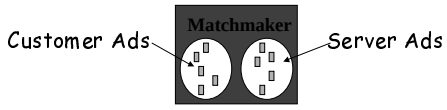
Example ClassAds (Cont.)

```
[
  Type = "Apartment";
  Location = "Downtown";
  HeatIncluded = true;
  Rent = 850;
  Phone = "608-555-1234";
  Requirements = other.Type == "Tenant" &&
        !other.HavePets && other.RentOffer >= Rent;
  Rank = other.RentOffer
]
```

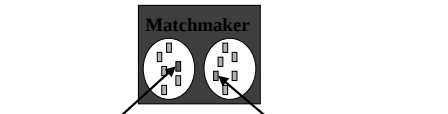

The Matchmaker

- Matchmaker matches compatible classads
  - Matchmaker ensures that Server and Customer Constraints are satisfied
  - Server and Customer Preferences are honored (within bounds of framework)
- Relevant parties notified when successfully matched
- Matched parties claim each other

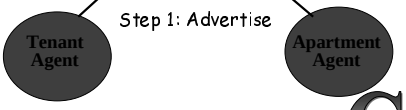
Matchmaking



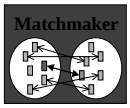
Matchmaking



Step 1: Advertise

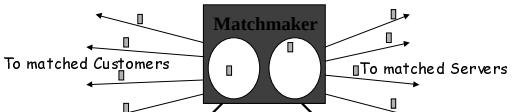


Matchmaking

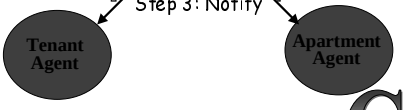


Step 2: Match

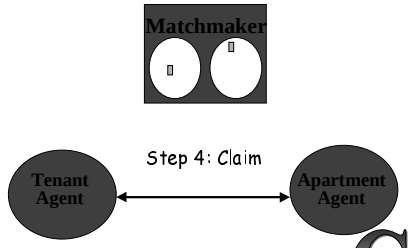
Matchmaking



Step 3: Notify



Matchmaking



Matchmaking Components

- Advertising Language
- Matchmaker Protocol
  - Defines how ads are sent to the Matchmaker
  - Names "well-known" attributes
  - Defines how matched entities are notified
- Matchmaking Algorithm
  - Algorithm used for making matches
- Claiming Protocol
  - Defines how matched entities establish an allocation

Classified Advertisements

- Set of (Attribute Name,Expression) pairs
- Self-describing (no separate schema)
  - Combines schema, data and query
  - Entities classads may include arbitrary attributes, so workstations may advertise
    - Physical location
    - Locally cached data files
    - Time windows when preemption is unlikely
- Syntax: [n0 = e0; n1 = e1; … ; ni = ei]
  - XML is a good fit; not currently used

Condor Examples

```
[ Type        = "Job";
  Owner       = "raman";
  Cmd         = "run_sim";
  Args        = "-Q 17 3200";
  Cwd         = "/u/raman";
  ImageSize   = 31M;
  Qdate       = 'Sat Jan 9... '
  ...
  Rank        = other.Kflops...
  Requirements =  ....
]
```

```
[ Type        = "Machine";
  Name        = "xxy.cs. ...";
  Arch        = "ALPHA";
  OpSys       = "OSF1";
  Mips        = 104;
  Kflops      = 21893;
  Memory      = 256M;
  LoadAvg     = 0.042969;
  ...
  Rank        = ...;
  Requirements = ...
]
```


Attribute Expressions

Constants	104, 0.042969, "iX86", true, '15:45', 'Sat Jan 9 15:33 1999 …'
References	attr, self.attr, other.attr
Operators	>>, <<, +, *,<, >=, &&, ...
Functions	strcat, substr, regexp, member,...
Lists	{ expr, expr, ... }
ClassAds	[name=expr; name=expr; ...]

Three-valued Logic

other.Memory > 32	all
other.Memory == 32	UNDEFINED
other.Memory != 32	if other has no
!(other.Memory == 32)	"Memory" attribute

Logical Operators (&& and ||) can squash UNDEFINED

For example:

other.Mips >= 100 || other.Kflops >= 1000

TRUE if *either* attribute exists and satisfies condition

Job Constraint Example

```
Requirements =
        other.Type = "Machine"
        && other.Arch = ”ALPHA"
        && other.OpSys = ”OSF1"
        && other.Disk > 1G
        && other.Memory >= self.ImageSize;
```


Rank Examples

```
    // job’s rank for machine
Rank = other.Kflops/1k + other.Memory/32M;


    // machine’s rank for job
Rank = member(other.Owner,ResearchGrp) ? 10
          : member(other.Owner, Friends) ? 5
          : 0;
```


Machine Constraint Example

```
IsIdle = LoadAvg<0.3 && KeyboardIdle>‘00:15’;

Requirements =
    !member(other.Owner,Untrusted) &&
  (
    Rank >= 10 ? True :
    IsIdle &&
        ( Rank >= 5 ? True : DayTime() < ‘6:00’ ||
        DayTime() > ‘18:00’ )
  )
```


Condor Matchmaking

- To match two ads A and B
  - Set up environment such that in A
    - self evaluates to A (MY in old classads)
    - other evaluates to B (TARGET in old classads)
- Check per classad policy
  - A.Requirements and B.Requirements
  - A.Rank and B.Rank for preferences
- Condor Matchmaking is a 4-way balancing act!
  - User Priorities, Job Preferences, Machine Preferences, Administrator Policies

Condor Matchmaking: Priorities

- Submitters are assigned priorities based on
  - Past resource usage
  - Administrator policy: boost and decay factors
- Submitters negotiated in priority order
  - Users with better priority more likely to get what they want
- Priorities used to determine "fair-share"
  - Share determined by ratio of priorities
  - May use more than fair-share if no one else wants resources

Condor Matchmaking: Preemption

- Job may be preempted for different reasons
  - Priority: New job's Owner has better priority
  - Rank: Machine likes new job more
- Administrator may control preemption
  - Stall priority preemption
    - PREEMPTION_REQUIREMENTS parameter
    - e.g., prevent "thrashing"
  - Affect choice of machine to preempt
    - PREEMPTION_RANK parameter
    - e.g., reduce network load

Condor Matchmaking: Scheme

- For submitter's job J (upto fair-share)
  - Only consider vacant or preemptable machines
  - Order lexicographically by
    1. Job rank of machine
    2. Preemption Mode
       (None > Rank > Priority)
    3. PREEMPTION_RANK (if applicable)
  - Check PREEMPTION_REQUIREMENTS if preempting for priority

Future Work

- Efficient Matchmaking
  - Attribute and constraint indexing schemes
  - Initial results very promising!
    - (8k,8k) takes 3 mins. instead of 2 hrs. 3 mins.
- Multilateral Matchmaking: Gang-Matching
  - Matchmaking with more than two entities
    - e.g., Job-Workstation-License
  - In addition to co-allocation, can implement
    - Regulation: limit number of running instances
    - Aggregation: abstract aggregate services

ClassAd Summary

- Distributed resource management
  - Distributed clients, servers
  - Heterogeneous resources
- Classified advertisements
  - Semi-structured data model
  - Schema, data, and query in one structure
- Matchmaking
  - Condor's matchmaking algorithm

ClassAd Summary (Cont.)

- ClassAds are used throughout Condor
  - Represent and query system daemons
    - startds, schedds, masters, collectors, checkpoint servers
  - Configuration and Job Control
- C++ and Java implementations
- Available as part of Condor and as stand-alone libraries