Resource Description and Selection

Andrew Chien
April 14, 2004
UCSD CSE225

Adminstrivia

- Homework #1 due monday 4/19/2004
- Initial Project Plans – Planning Meetings Tomorrow and Friday
- No CSE225 on Wednesday, April 21
Initial Project Plans

- Design, Implementation, and Evaluation of Randomized Heuristic Selection Algorithms for Grid Resource Selection (Richard Huang, Nut Taesombut, Erik Vandekieft)
- Resource Sharing and Application Benchmarks (Catherine Olschanowsky, Jacob Sorensen, Eric Weigle)
- An Application-driven Evaluation of Grid Infrastructures (Jonathan Weinberg, Apurva Sharma, Jiahua He, Kiran Kalyan)
- Scalable Resource Discovery for Large-Scale Federated Systems (Jeannie Albrecht)

Last Time: Resource Discovery and Management

- Resource Discovery
  » MDS, Grid Information Services
- Resource Management
  » Application View and System View
  » RSL: a language to describe resource needs
  » GRAM, co-allocators, brokers: services to gain access to resources
- Javaspaces/Jini: Another Grid Infrastructure
  » Idea of Tuple Spaces and Generic Matching
  » Javaspaces and Additional Jini infrastructure (leases, transactions, etc.)
  » Services as the Basis of Advertisement
Today’s Readings

- **UDDI**:  
  » Two Overviews (IBM)  
  » Scenarios Document (Microsoft)  
  » uddi.org, UDDI Full spec (400+ pages, don’t need to read this)

- **A Constraint Language Approach to Grid Resource Selection** (Liu and Foster), Tech Report  
  (Raman, Livny and Solomon), Condor Matchmaking

UDDI: Universal Description, Discovery, and Integration

- A long-running effort to enable service integration  
  » Within enterprises  
  » Across enterprises  
  » Recent focus on web services

- Version 1, Version 2 -- 2001
- Version 3 – 2003?
UDDI Basic Architecture

UDDI and SOAP

<table>
<thead>
<tr>
<th>Client</th>
<th>UDDI SOAP request</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UDDI SOAP response</td>
</tr>
</tbody>
</table>

Create, view, update, and delete registrations
Implementation neutral

UDDI registry node

HTTP server → SOAP server

Process UDDI API request

Registry data

UDDI Role in Web Services

WSIFL

Service flow

Service discovery

Service publication

Security

WSIDL

Service description

Management

SOAP

XML-based messaging

Quality of service

http, ftp, IM, #OR and more

Network
Business Use Model

Businesses consult the registry to facilitate easier integration with each other over the Web.

UDC registry assigns a programmatically unique identifier (UDCID) to each tModel and business registration and stores an internal registry.

Inquiry Operations:
- Find
  - find_business
  - find_service
  - find_binding
  - find_tModel
- Get details
  - get_business
  - get_serviceDetail
  - get_bindingDetail
  - get_tModelDetail
  - get_registeredInfo

Publishing Operations:
- Save
  - save_business
  - save_service
  - save_binding
  - save_tModel
- Delete
  - delete_business
  - delete_service
  - delete_binding
  - delete_tModel
- Security
  - get_authToken
  - discard_authToken
Web Services and UDDI Types

Mapping from WSDL to UDDI

Example Use

• Search for a relevant Service Registration
  » find_business, find_service, find_binding, find_tModel
• Get Detail
  » get_business, get_serviceDetail, get_bindingDetail, get_tModelDetail, get_registeredInfo
• Use the services
  » Make invocations to them
Discussion: Relation to Grids

- Distributed Service Location
- Access to Resources
- Scaling
- How Dynamic are the expected bindings?

Matchmaking and Redline

- Matchmaking in Condor as a background
  » Extensions to Sets, Gangmatching
- Redline as an Improvement
Original Condor Model (1986)

- Cycle harvesting – Maximize resource utilization
- Checkpoint and Migrate – ensure completion eventually
- Respect the rights of resource owners (unobtrusiveness)
- Efficient: moderate overhead in use of resources

=> Resource selection and binding problem is different from typical
  » Complex set of resource conditions for use
  » Per resource policies; Extreme heterogeneity (policy and hardware)
  » Selection repeatedly; at each migration

Running a Condor Job

- Describe Job requirements in ClassAd
- Job Submitted to system
- Find a resource that’s appropriate (selection?)
- Claim that resource and run (binding?)
- Return results and complete
  » Many intermediate changes/interruptions…
Matchmaking

- ClassAds and suitable resources
  - Allowable use: cpu, memory, load factor, allowable user
  - Job requirements: cpu, memory, load factor, etc.

- Constraint $\leftrightarrow$ Compatibility
- Rank $\leftrightarrow$ Goodness Metric

- Matchmaker accepts Ads then introduces “matches”
- $\Rightarrow$ selection = introduction
- $\Rightarrow$ binding achieved by pairwise contact

What Are ClassAds?

- A ClassAd maps attributes to expressions
- Expressions
  - Constants: strings, numbers, etc.
  - Expressions: other. Memory > 600M
  - Lists: { “roy”, “pfc”, “melski” }
  - Other ClassAds
- Powerful tool for grid computing
  - Semi-structured (you pick your structure)
  - Matchmaking
ClassAd Example

```plaintext
[  Type = "Job";
   Owner = "roy";
   Universe = "Standard";
   Requirements = (other.OpSys == "Linux" && other.DiskSpace > 140M);
   Rank = (other.DiskSpace > 300M ? 10 : 1);
   ClusterID = 12314;
   JobID = 0;
   Env = "";
   ...
]
```

ClassAd Matchmaking

```plaintext
[  Type = "Job";
   Owner = "roy";
   Requirements = (other.OpSys == "Linux" && other.DiskSpace > 140M);
   Rank = (other.DiskSpace > 300M ? 10 : 1);
]

[  Type = "Machine";
   OpSys = "Linux";
   DiskSpace = 500M;
   AllowedUsers = {"roy", "melski", "pfc"};
   Requirements = (IsMember(other.Owner, AllowedUsers);
]
```
Claiming a Resource

- **Bilateral matching**
  - No “client or server”
- **Ensure pairing is mutually acceptable**
  - All of the dynamic requirements are met (at least for now)
  - Additional static requirements are met (e.g. authentication)
  - ANY other protocol that you might want to enforce
  - RANK indicates preference

- If matching fails, return to the matchmaker
  - Another Ad or update availability of an Ad

Unobtrusiveness

- Allowable use defined for each resource (ClassAd)
  - Load factor (cpu)
  - Free memory (memory)
  - Keyboard Idle (user)
  - Allowed users (?)
- => jobs only scheduled when the conditions are met

- These conditions may change -- jobs are checkpointed locally when conditions are violated
- Job will be suspended until the conditions are acceptable
Checkpointing / Execution Model

• Applications run in a “virtual environment” of the submitter’s machine
  » Remote I/O to user’s machine
  » Standard I/O to user
• Applications are long-running
  » May surpass the “availability window” of a resource
  » Desire to conserve the work invested in the application
• Checkpoint when application is interrupted
  » Change of local conditions – resource no longer available
  » Another application preempts – takes over the machine
  » Can be resumed on machine of same architecture

Matchmaking and Extensions

Matchmaking Extensions

• GangMatching
  » Allows a class ad to include list of “ports” (each really a class ad with some scoping for shared names)
  » A range of indexing and optimization techniques for implementation
  » Matchmaker generates a “gang match” introduction
• Set Matching
  » Allows expressions over aggregates (min, max, sum)
  » Returns a set of class ads which meet the desired aggregate properties (and individual properties)
A Constraint-based Approach

- Redline Language
  - Define resource request as a set unbound variables (resource names) and constraints on them
  - Define resource attributes as a set of constraints
- Match = binding of the resource names to specific resources
- Leverage wealth of investment and knowledge about constraint languages and solvers

Redline Constraint Language

- Types=(real,int,string,boolean,UNDEF,ERR)
- Description= constraints*
- Constraints= variable op expression
  - Expression = variable op variable | pred expression | value
- Predicates: minimize, maximize, forall X in SET, forany X in SET, required
- Sets, Lists, Enumerates
- Set Operations: count, max, min, sum, inSet, set_intersect, set_union, set_difference, set_s_difference
- => Full-blown constraint language, with range of intermediate variables and terms
Redline Examples

```plaintext
[set user="globus-user"; group="dsl-uc";
computation isa
  set [type="computation"];
storage isa [type="storage";
  space>100];
forall x in computation;
x.cpuspeed>150;
x.bandwidth[storage.hn]>30;
x.accesstime>80;
sum(computation.memory)>300;
storage.space>80;
storage.accesstime>18]
```

```plaintext
R1=[type="computation";
  hn="ucsd1"; cpuspeed=200;
  bandwidth=DICT[{"s1",20},{"s2",40}];
  accesstime>17]
R2=[type="computation";
  hn="ucsd2"; cpuspeed=200;
  bandwidth=DICT[{"s1",20},{"s2",40}];
  accesstime>17]
R3=[type="storage"; hn="s1"; space=100]
R4=[type="storage"; hn="s2"; space=200]
```

Benefits

- **Tremendous Leverage**
  - Well understood language and solver properties
  - Well developed implementations and solvers
- **Regular framework is easily extensible**
- **Full constraint language is VERY powerful**
  - All of the things in other languages
  - General queries, incremental changes
  - Full symmetry and enumeration
Challenges

- Performance
  » Cost of constraint solver, realistic resource pool sizes and structures
  » Centralized structure, how to deal with distributed information – can preserve properties?
- Complexity
  » Richness of interface complicates specification
  » Do we want very sophisticated, complex resource descriptions?
- Semantics and Properties over Dynamic Information
  » What properties can we assure?
- Others?

Discussion

- Resource Selection
- Complexity of Resource Descriptions
- Dynamic Performance Information
- Selection vs. Binding
Summary

• UDDI
  » Describe, Save, Modify Mechanisms
  » Implementation Architecture
  » Integration with Web Services
• Matchmaking
  » Basic Condor
  » Extensions: Gangmatching, Set Matching
  » Redline: Generalized Constraint Matching
• Next time: Resource Requirement Descriptions and Resource Utilization