Ajents: Towards an Environment for Parallel, Distributed and Mobile Java Applications

Matthew Izatt, Patrick Chan
York University

Tim Brecht
University of Waterloo

http://www.cs.uwaterloo.ca/~brecht
AJENTS: GOALS

- Simplify writing and running distributed applications (no account required on serving machines)
- Seamless access to heterogeneous resources on network
- Parallel, distributed and mobile Java objects
- Serial jobs remotely (faster machines / more memory)

- World-Wide Computing
- Leverage existing Java technology
- JAVA COMPATIBILITY
AJENTS: DESIGN

- Java class libraries (standard and portable)
- Remote object creation (only need byte-code)
- Synchronous and asynchronous RMI
- Object migration
SUN RMI: REMOTE OBJECT

public interface B
extends java.rmi.Remote
{
    String getName()
    throws java.rmi.RemoteException;
    String setName(String name)
    throws java.rmi.RemoteException;
}

public class BImpl extends UnicastRemote
    implements B
{
    ...

prompt> rmic BImpl
public class B implements Serializable // to migrate
{
    String name;

    public B() {
        name = new String("Default");
    }

    public String getName() {
        return name;
    }

    public String setName(String name) {
        String oldname = this.name;
        this.name = name;
        return oldname;
    }
}
REMOTE OBJECT CREATION

SUN

- Find available host, install code and run object ("B1")
- At client side obtain reference to B1

\[ B \text{ bl} = (B) \text{ Naming.lookup("rmi://Host2/B1");} \]

AJENTS

- Admins run Ajents server & sched on remote hosts
- At client side create remote objects

\[
\text{AjentsSched } s = \text{ Ajents.register();}
\text{AjentsObj bl = Ajents.new("ajents.B", "B1", "/ajents/myobjs.jar", s.AvailServer());}
\]
AJENTS: REMOTE OBJECT CREATION

Scheduling Host

Main Pgm

AjetsObj

RObj ref

Originating Host

Sched Server

Ajets Server

Remote Host

RObject

B1 (created)
SUN RMI

try {
    oldname = b1.setName("NewB1");
    name = b1.getName();
} catch (Exception ex) { ... }

AJENTS RMI

try {
    oldname = Ajents.rmi(b1, "setName", "NewB1");
} catch (Exception ex) { ... }

Future f = Ajents.armi(b1, "getName");

try {
    oldname = (String) f.get();
} catch (Exception ex) { ... }
AJENTS: OBJECT MIGRATION

• By the creating object, Ajents server, or 3rd party

1) Immediate migration of idle objects
2) Delayed migration (until idle)
3) Immediate migration using checkpointing and roll back

```java
public int method(int N) {
    for (int i=0; i<N; i++) {
        // do something
        . . .
    }
    return value;
}
```
Ajents.migrate(b1, sched.AvailServer());

// Turn checkpointing on
Ajents.setCheckPoint(b1, true);

// b1 checkpointed, method invoked
Future f = Ajents.armi(b1, "getName");

// move b1, possibly before armi finishes
Ajents.migrate(b1, sched.AvailServer());

name = (String) f.get();
AJENTS: OBJECT MIGRATION

Host A
AjentsObj
(1) RMI

Host B
Ajents Server Obj1 (migrated)

Host C
Ajents Server Obj1
(2) new reference
(3) RMI (second try)
## MICRO-BENCHMARK PERFORMANCE

### RMI

<table>
<thead>
<tr>
<th>Object Size</th>
<th>(ints)</th>
<th>1</th>
<th>1k</th>
<th>10k</th>
<th>100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMI - Ajents</td>
<td>(ms)</td>
<td>9</td>
<td>11</td>
<td>46</td>
<td>400</td>
</tr>
<tr>
<td>RMI - JDK RMI</td>
<td>(ms)</td>
<td>5</td>
<td>8</td>
<td>44</td>
<td>400</td>
</tr>
</tbody>
</table>

### MIGRATION

<table>
<thead>
<tr>
<th>Object Size</th>
<th>(ints)</th>
<th>1</th>
<th>1k</th>
<th>10k</th>
<th>100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migration time</td>
<td>(ms)</td>
<td>334</td>
<td>334</td>
<td>397</td>
<td>769</td>
</tr>
</tbody>
</table>

### CHECKPOINTING

<table>
<thead>
<tr>
<th>Object Size</th>
<th>(ints)</th>
<th>1</th>
<th>1k</th>
<th>10k</th>
<th>100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-Checkpointing</td>
<td>(ms)</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Checkpointing</td>
<td>(ms)</td>
<td>8</td>
<td>9</td>
<td>19</td>
<td>115</td>
</tr>
<tr>
<td>Checkpointing Cost</td>
<td>(ms)</td>
<td>2</td>
<td>3</td>
<td>13</td>
<td>109</td>
</tr>
</tbody>
</table>
EXAMPLE APPLICATION MIGRATION OVERHEADS

\[
\begin{bmatrix} A \times B = C \end{bmatrix}
\]

\[
\begin{bmatrix} A \times B = C \end{bmatrix}
\]

\[
\begin{bmatrix} A \times B = C \end{bmatrix}
\]

\[
\begin{bmatrix} A \times B = C \end{bmatrix}
\]

<table>
<thead>
<tr>
<th>N</th>
<th>200</th>
<th>500</th>
<th>640</th>
<th>800</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migrated</td>
<td>0.5</td>
<td>2.9</td>
<td>4.7</td>
<td>7.3</td>
<td>12.0</td>
</tr>
<tr>
<td>1 Server</td>
<td>7.7</td>
<td>134</td>
<td>292</td>
<td>580</td>
<td>1305</td>
</tr>
<tr>
<td>8 Servers</td>
<td>20.3</td>
<td>165</td>
<td>344</td>
<td>666</td>
<td>1402</td>
</tr>
<tr>
<td>Inflation</td>
<td>2.64</td>
<td>1.23</td>
<td>1.18</td>
<td>1.15</td>
<td>1.07</td>
</tr>
</tbody>
</table>

Times in seconds, migrated data in MB.

Inflation = time on 8 servers / time on 1 server.
## PARALLEL APPLICATION PERFORMANCE

<table>
<thead>
<tr>
<th>N</th>
<th>200</th>
<th>500</th>
<th>640</th>
<th>800</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq. C</td>
<td>4.0</td>
<td>67</td>
<td>139</td>
<td>280</td>
<td>601</td>
</tr>
<tr>
<td>Seq. Java</td>
<td>7.5</td>
<td>131</td>
<td>286</td>
<td>574</td>
<td>1272</td>
</tr>
<tr>
<td>1 Server</td>
<td>7.7</td>
<td>134</td>
<td>292</td>
<td>580</td>
<td>1305</td>
</tr>
<tr>
<td>8 Servers</td>
<td>1.5</td>
<td>20</td>
<td>40</td>
<td>78</td>
<td>172</td>
</tr>
<tr>
<td>Speedup</td>
<td>5.1</td>
<td>6.7</td>
<td>7.3</td>
<td>7.4</td>
<td>7.6</td>
</tr>
</tbody>
</table>

Time in seconds, speedup relative to Seq. Java

(Java Grande ’99)
RELATED WORK

- ABC++  
  [Arjomandi et al. 95]
- ParaWeb  
  [Brecht et al. 96]
- Mole  
  [Straßer et al. 96]
- SuperWeb  
  [Alexandrov et al. 97]
- Javelin  
  [Christiansen et al. 97]
- JavaParty  
  [Philippsen & Zenger 97]
- Asynchronous RMI  
  [Raje et al. 97]
- Aglets  
  [Lange & Oshima 98]
- Fast RMI  
  [Krishnaswamy et al. 98]
  
  [Maassen et al. 99]

- Many others
CONCLUSIONS

• Java Compatible (pros and cons)
• Remote object creation (only need byte-code)
• Synchronous or asynchronous remote method invocation
• Object migration

• Quite reasonable performance
• Simple to build applications with good performance

http://www.cs.uwaterloo.ca/~brecht