Efficient Replicated Method Invocation in Java

Vrije Universiteit
Amsterdam
Faculty of Sciences

Jason Maassen
Henri Bal
Thilo Kielmann
Introduction

• Parallel programming in Java requires:
  – efficient execution of Java code
  – efficient communication (Remote Method Invocation)

  – (experience with Orca) efficient object replication

• Manta system [PPoPP ‘99, JavaGrande ‘99]
  – native compilation of Java
  – fast RMI system (both Sun and JavaParty model)

• Next step: extend Manta with replication
Replication Schemes

- Invalidation based (data shipping)
  - delete copies on write
  - inefficient for large data structures

- Update based (function shipping)
  - update copies on write by forwarding the method invocation

- Experience with Orca
  - function shipping most efficient for object based languages

- Function shipping replication as extension of RMI
Replication in Manta

- Replicated object is a special Remote object
  - both use function shipping

- Remote objects:
  - method invocations are forwarded to single object
  - no distinction between read/write methods

- Replicated objects:
  - method invocations are forwarded to multiple objects
    (use totally-ordered multicast)
  - write methods forwarded / read methods locally
  - high read / write ratio reduces communication overhead
Replicating Object Graphs (1)

- Java object model:
  - many small objects linked by references

- A method can affect a graph of objects
  - by following references and invoking other methods

```
List list1 = new List();
list1.m1();
List head = list1.head();
head.m2();
```
Replicating Object Graphs (2)

• Replicating without references is restrictive
  – can only replicate objects containing primitive types

• Replicating separately is incorrect
  – causes the nested invocation problem

• Solution
  – replicate a graph of objects as a single entity, a cluster
Clusters

• Closed graph of objects
  – no reference into or out of cluster

• Single entry point: root object
  – only object that can be accessed from outside the cluster

• Objects reachable from root: node objects
  – cannot be referenced from outside the cluster

• Method invocations on root broadcast to all copies
Cluster example
Programming interface

• Tag objects using “special” interfaces
  – recognized by the Manta compiler
  – similar to “Serializable” or “Remote”
    – manta.replication.Root
    – manta.replication.Node

• No language extensions
class Stack implements manta.replication.Root {

    StackNode top = null;

    void push(int value) { … } // write operation

    int pop() { … } // write operation

    int top() { … } // read operation

}

class StackNode implements manta.replication.Node {

    StackNode prev;
    int value;

    StackNode(int value, StackNode prev) { … }

}
Restrictions

• Similar to RMI:
  – no direct access to fields from outside
  – use of static variables is dangerous
  – call-by-value

• To enforce clustering model:
  – cluster objects may only reference Node objects
  – cluster objects may only implement one “special” interface
  – only call well behaved methods
Implementation

• Read / write analysis
  – done by compiler
  – problems with polymorphism
  – final decision at runtime

• Synchronization (wait / notify)
  – must maintain global ordering of write methods
  – no order in Java threads (especially across machines)
  – in Manta threads are replication aware
    (wake up in same order on all machines!)
Performance

- Benchmark (replicated stack, in micro seconds)
  - Pentium Pro 200, Myrinet network + efficient multicast

<table>
<thead>
<tr>
<th></th>
<th>cpus</th>
<th>push</th>
<th>top</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential</td>
<td></td>
<td>3.1</td>
<td>0.1</td>
</tr>
<tr>
<td>RMI, local</td>
<td></td>
<td>6.1</td>
<td>2.7</td>
</tr>
<tr>
<td>RMI, remote</td>
<td></td>
<td>55.3</td>
<td>49.2</td>
</tr>
<tr>
<td>replicated</td>
<td>1</td>
<td>29.1</td>
<td>0.5</td>
</tr>
<tr>
<td>replicated</td>
<td>2</td>
<td>60.0</td>
<td>0.5</td>
</tr>
<tr>
<td>replicated</td>
<td>4</td>
<td>62.3</td>
<td>0.5</td>
</tr>
<tr>
<td>replicated</td>
<td>8</td>
<td>65.3</td>
<td>0.5</td>
</tr>
<tr>
<td>replicated</td>
<td>16</td>
<td>68.0</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Traveling Sales Person Speedup

- TSP with shared minimum
All Pairs Shortest Path Speedup

- ASP with shared matrix
Conclusions

• Replication in Manta:
  – clean model (extension of Remote Method Invocation)
  – replicates clusters (closed graphs of objects)
  – normal Java synchronization allowed inside cluster
  – efficient implementation
Future work

- How restrictive is the model?
  - test more applications (already have: LEQ, QR, ACP)

- Test applications on our wide area system

- Extend with object migration
  - decide at runtime to replicate/migrate a cluster

- http://www.cs.vu.nl/manta
Distributed ASCI Supercomputer (DAS)

- A 200-node system consisting of 4 Myrinet-based Pentium Pro clusters

- Each node:
  - 200 MHz Pentium Pro
  - 128 MB EDO-RAM in DIMM modules
  - 2.5 GByte local disk
  - Myrinet interface card
  - Fast Ethernet interface card

- http://www.cs.vu.nl/das/
<table>
<thead>
<tr>
<th>Orca</th>
<th>Manta</th>
</tr>
</thead>
<tbody>
<tr>
<td>language designed for replication</td>
<td>general language</td>
</tr>
<tr>
<td>no references between objects</td>
<td>inter-object references</td>
</tr>
<tr>
<td>methods access a single object</td>
<td>methods can access multiple objects</td>
</tr>
<tr>
<td>no global data</td>
<td>global (static) variables</td>
</tr>
<tr>
<td>methods only block at start</td>
<td>methods can block at any time</td>
</tr>
</tbody>
</table>
Does this work on JDK?

- extra analysis / code generation (no problem)
  - read / write analysis
  - method wrappers

- need totally-ordered multicast support

- correct wait / notify implementation
  - requires control over thread ordering
  - problem: wait / notify cannot be overloaded
Arc Consistency Problem Speedup

- ACP with shared matrix

![Graph showing speedup vs cpus for ACP 2000+150 with different optimization levels: perfect, replicated, manually optimized, naive.]
QR Calculation Speedup

- QR with shared "broadcast" object.

![QR Calculation Speedup Graph](QR_2000x2000.png)