Fault-tolerant data sharing for high-level grid programming: a hierarchical storage architecture

Marco Aldinucci, Marco Danelutto
Dept. of Computer Science, University of Pisa, Italy

Gabriel Antoniu, Mathieu Jan
INRIA Rennes, France
Marian’s exercise ...

UniPisa

Grid-enabled high-level programming model (with data sharing)

INRIA Rennes

Robust data sharing service for the grid
Marian's exercise ...

ASSIST with its cluster-oriented sharing service

JuxMem Jxta-based fault-tolerant sharing service

Memory hierarchy transparently supporting grid-level coherent, fault-tolerant, persistent data sharing. First prototype supports data sharing in ASSIST applications.
Marian’s exercise ...

ASSIST with its cluster-oriented sharing service

Memory hierarchy transparently supporting grid-level coherent, fault-tolerant, persistent data sharing. First prototype supports data sharing in ASSIST applications.

... while waiting for the “Philosophy of the Grid”

Not innovative?
At the bottom line, Grid appears more evolutionary than revolutionary, isn’t it?
The two software tools
- ASSIST (high-level programming model)
- JuxMem (grid data service)
- exploit their complementarity

How they have been integrated
- a memory hierarchy, with locality

Prototype, experiments (preliminary)
Data management in grid

- Memory storage features
  - Persistency (survive to application instances)
  - Robustness (fault-tolerance)
  - Efficiency (not only in ftp, but real RAM storage)
- In high-level programming models
  - Transparent access from programming model
  - Run-time supp. implementation (e.g. FT message logs)
High-level programming model
Based on parallel modules
  - GCM components ongoing
Modules exchange data via streams and/or shared memory
  - sharing implemented via distributed memory server (called ASSIST/ad-HOC)
  - read, write (in parallel) distributed “objects” identified by a logical ID
app = graph of modules

input

P1 → P2 → P3 → P4

output
app = graph of modules

input

P1 → P2 → P3 → P4

output
app = graph of modules

Sequential or parallel module
(native or wrap e.g. MPI, CCM)
app = graph of modules

Programmable, possibly nondeterministic input behavior

Sequential or parallel module (native or wrap e.g. MPI, CCM)

Typed streams of data items (TCP/IP, Globus, IIOP CORBA, HTTP/SOAP)
Supports data sharing

P1 → P2 → P3 → P4

ad-HOC distr. data component
ad-HOC distr. data component
Supports data sharing

1. Shared state within a parmod (attributes)
Supports data sharing

1. Shared state within a parmod (attributes)

ad-HOC distr. data component

ad-HOC distr. data component
Supports data sharing

1. Shared state within a parmod (attributes)
2. Shared state among parmods (references)
• Grid data service
  • P2P JXTA-based prototype
  • Transparent access to data blocks
  • Persistent storage
  • Mutable data: consistency guarantees
  • Active support for peer volatility

• API
  • alloc, map, get, put, lock, unlock
Overview of JuxMem’s architecture

User should define:
1. on how many cluster replicate data;
2. how many providers in each cluster,
3. consistency protocol
Overview of JuxMem’s architecture

- User should define:
  1. on how many cluster replicate data;
  2. how many providers in each cluster,
  3. consistency protocol
JuxMem core layers used to test P2P techniques over grid infrastructures

- **JuxMem core (juk)**
  - Memory storage
  - Communication
  - Discovery

- **Fault tolerance protocols**
  - Consistency protocols
  - Group membership
  - Atomic multicast
  - Consensus
  - Failure detector

**JuxMem layers**
**Comparison at hand**

No free lunches in nature ...

<table>
<thead>
<tr>
<th></th>
<th>Cluster sharing (ad-HOC)</th>
<th>Grid sharing (JuxMem)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Throughput</strong></td>
<td>High</td>
<td>High-Medium</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>Parallel access to a single data item</strong></td>
<td>Read/Write</td>
<td>Read only</td>
</tr>
<tr>
<td><strong>Data consistency</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Fault-tolerance (data replication)</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Dynamically reconfigurable</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Data location transparency</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
JuxMem and ad-HOC can be organized in a two-tier memory hierarchy

- robust *almost* as JuxMem
- fast *almost* as ad-HOC

provided that

- data locality is promoted by the programming model
- data is transparently exchanged between the two tiers
Integrated architecture

**Memory Hierarchy**

- **grid-level storage & data service**
  (JuxMem-based: distributed, permanent, fault-tolerant)

- **cluster-level storage**
  (ASSIST-based: parallel, distributed)

- **stream connections**

- **access speed, frequency, & locality**

- **robustness persistence access grain**

- **cluster tier (ASSIST/ad-HOC)**

- **grid tier (Juxmem)**
Data locality

- Two kinds
  - classical spatial/temporal locality
  - clustered locality
- Enforced by programming model
  - skeletons/paradigms lead to regular interaction patterns
  - modules/components helps to enforce locality delimitating activities with frequent interactions
    - Fractal, GCM & hierarchic models (provided a proper mapping exists)
Useful for what?

- Sharing across multiple clusters
  - sharing among different applications (persistency)
  - relax co-allocation constraints via stream buffering
    - Direct Acyclic Graphs does not need strict co-allocation
  - data is stored in safe w.r.t. node faults
- Fault-tolerant data storage/checkpointing
  - checkpointing driven by app semantics
  - ASSIST already instruments apps with reconf-safe points for adaptivity (data on ad-HOC is “coherent”)
Prototype & experiments

- A preliminary prototype exists
  - developed by students (master thesis)
    - software engineering time not fully predictable
    - CoreGRID does not pay SW engineers
- Experiments are also preliminary
  - focused on the correctness of the system
  - focused on behavior of parts “in insulation”
    - some examples follows
A preliminary prototype exists
- developed by students (master thesis)
  - software engineering time not fully predictable
  - CoreGRID does not pay SW engineers

Experiments are also preliminary
- focused on the correctness of the system
- focused on behavior of parts “in insulation”
  - some examples follows
mediators are triggered by application processes (compiler instrumented)
they read/write data between the two memory tiers
JuxMem bandwidth (G-Eth)
JuxMem bandwidth (G-Eth)
- Good BW with no replicas
- Latency not excellent
- \( BW = O\left(\frac{K}{\text{#replicas}}\right) \)
- Hierarchy aims to reduce the frequency of the access
<table>
<thead>
<tr>
<th>Arch/Net/OS</th>
<th>concurrent connections</th>
<th>Msg size (Bytes)</th>
<th>Replies/Sec</th>
<th>net throughput (Bytes/Sec)</th>
<th>net throughput w.r.t. ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4@2GHz</td>
<td>2048</td>
<td>1 M</td>
<td>91</td>
<td>91 M</td>
<td>96%</td>
</tr>
<tr>
<td>Mem 512MB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GigaEth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux ker. 2.4.22</td>
<td>3072</td>
<td>512</td>
<td>20 M</td>
<td>10 M</td>
<td>11%</td>
</tr>
<tr>
<td>P3@800MHz</td>
<td>1024</td>
<td>8 K</td>
<td>1429</td>
<td>11.2 M</td>
<td>90%</td>
</tr>
<tr>
<td>Mem 1GB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FastEth</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux ker. 2.4.18</td>
<td>1024</td>
<td>16 K</td>
<td>718</td>
<td>11.2 M</td>
<td>90%</td>
</tr>
</tbody>
</table>
## ad-HOC figures

<table>
<thead>
<tr>
<th>Arch/Net/OS</th>
<th>concurrent connections</th>
<th>Msg size (Bytes)</th>
<th>Replies/Sec</th>
<th>net throughput (Bytes/Sec)</th>
<th>net throughput w.r.t. ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td>P4@2GHz Mem 512MB GigaEth Linux ker. 2.4.22</td>
<td>2048</td>
<td>1 M</td>
<td>91</td>
<td>91 M</td>
<td>96%</td>
</tr>
<tr>
<td></td>
<td>3072</td>
<td>512</td>
<td>20 M</td>
<td>10 M</td>
<td>11%</td>
</tr>
<tr>
<td>P3@800MHz Mem 1GB FastEth Linux ker. 2.4.18</td>
<td>1024</td>
<td>8 K</td>
<td>1429</td>
<td>11.2 M</td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>16 K</td>
<td>718</td>
<td>11.2 M</td>
<td>90%</td>
</tr>
</tbody>
</table>

+ 7%
## ad-HOC figures

<table>
<thead>
<tr>
<th>Arch/Net/OS</th>
<th>concurrent connections</th>
<th>Msg size (Bytes)</th>
<th>Replies/Sec</th>
<th>net throughput (Bytes/Sec)</th>
<th>net throughput w.r.t. ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P4@2GHz</strong>&lt;br&gt;Mem 512MB&lt;br&gt;GigaEth&lt;br&gt;Linux ker. 2.4.22</td>
<td>2048</td>
<td>1 M</td>
<td>91</td>
<td>91 M</td>
<td>96%</td>
</tr>
<tr>
<td></td>
<td>3072</td>
<td>512</td>
<td>20 M</td>
<td>10 M</td>
<td>11%</td>
</tr>
<tr>
<td><strong>P3@800MHz</strong>&lt;br&gt;Mem 1GB&lt;br&gt;FastEth&lt;br&gt;Linux ker. 2.4.18</td>
<td>1024</td>
<td>8 K</td>
<td>1429</td>
<td>11.2 M</td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>16 K</td>
<td>718</td>
<td>11.2 M</td>
<td>90%</td>
</tr>
</tbody>
</table>

+ 7%

+ ~1000%
ad-HOC figures

- Good BW for small data size
- Good latency (not shown)
- Good support for concurrent connection (firewalls & co)
- No data replication, no FT, cluster-oriented

<table>
<thead>
<tr>
<th>Arch/Net/OS</th>
<th>Msg size (Bytes)</th>
<th>Replies/Sec</th>
<th>net throughput (Bytes/Sec)</th>
<th>net throughput w.r.t. ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gigabit, Linux ker. 2.4.22</td>
<td>1 M</td>
<td>91</td>
<td>91 M</td>
<td>96%</td>
</tr>
<tr>
<td></td>
<td>3072</td>
<td>512</td>
<td>20 M</td>
<td>11%</td>
</tr>
<tr>
<td>P3@800MHz, Mem 1GB, FastEth, Linux ker. 2.4.18</td>
<td>1024</td>
<td>8 K</td>
<td>1429</td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>16 K</td>
<td>718</td>
<td>90%</td>
</tr>
</tbody>
</table>
Conclusions

- Both JuxMem (F) and ASSIST/ad-HOC (I) implement data storage services.
  - They exhibit a similar API but complementary aims and features.

- They can be composed to set up a parallel, distributed, efficient, fault-tolerant memory hierarchy
  - Real integration of existing (and complex) software developed by different CoreGRID partners
  - It enables the experimentation of architectural solution for high-performance robust data services for the grid
  - It can be used as robust storage for checkpoints

- Aims to understanding how memory hierarchies work in grid env., and how they are related to programming model (beyond ASSIST ...)