Adapting Distributed Shared Memory Applications in Diverse Environments

Daniel Potts and Ihor Kuz

danielp@cse.unsw.edu.au

University of New South Wales, Sydney, Australia

and

National ICT Australia
Overview

• Motivation
• Related work
• View model
• Experiments and Results
Application:

A Matrix Multiply implemented using Lazy Release Consistency (LRC) for a cluster of Linux nodes with Ethernet interconnect.
Motivation: Diverse Environments

Problem: Computation environments are diverse.
Motivation: Diverse Environments

Problem: Computation environments are diverse.

- Poor resource utilisation
Motivation: Diverse Environments

Problem: Computation environments are diverse.

- Poor resource utilisation
- Environment structure ignored
Motivation: Diverse Environments

Problem: Computation environments are diverse.

- Poor resource utilisation
- Environment structure ignored
- Heterogeneous environments poorly supported
Motivation: Diverse Environments

Problem: Computation environments are diverse.

- Poor resource utilisation
- Environment structure ignored
- Heterogeneous environments poorly supported
- Wide-area poorly supported
Goals

• Run-time adaption to different homogeneous environments

• Optimise for environment structure

• Utilise resources of heterogeneous environments
Related Work (Existing Solutions)

• Run-time adaption to different homogeneous environments:
  ⇒ Protocol selection in DSM-PM2
• Optimise for environment structure:

• Utilise resources of heterogeneous environments:
Related Work (Existing Solutions)

• Run-time adaption to different homogeneous environments:
  ⇒ Protocol selection in DSM-PM2
• Optimise for environment structure:
  ⇒ Home-based protocols eg. Home-based LRC
  ⇒ Hybrid protocols eg. Albatross
• Utilise resources of heterogeneous environments:
Related Work (Existing Solutions)

- Run-time adaptation to different homogeneous environments:
  - Protocol selection in DSM-PM2
- Optimise for environment structure:
  - Home-based protocols eg. Home-based LRC
  - Hybrid protocols eg. Albatross
- Utilise resources of heterogeneous environments:
  - Poor performing generic software protocols
Related Work (Existing Solutions)

- Run-time adaption to different homogeneous environments:
  - Protocol selection in DSM-PM2

- Optimise for environment structure:
  - Home-based protocols eg. Home-based LRC
  - Hybrid protocols eg. Albatross

- Utilise resources of heterogeneous environments:
  - Poor performing generic software protocols

\[ \times \text{No overall solution} \]
Can we develop a flexible model to meet goals?
Views: An abstraction for protocol encapsulation

- The view model separates:
  - programming model,
  - consistency protocol,
  - communication protocol,
  - sharing interactions,
  - execution environments,

  to give us *flexibility*. 
Approach using Views

- **Green** view: single protocol, identical to traditional approach
Approach using Views

- **Green** view: single protocol, identical to traditional approach
- **Purple** view: data access localised to clusters
Approach using Views

- **Green** view: single protocol, identical to traditional approach
- **Purple** view: data access localised to clusters
- **Pink/Blue** view: use optimised protocols
Views: Non-Overlapping

- View clients (e.g. C1) represent data sharers such as threads.
- An application may utilise many views.
- Can use different data sharing semantics for different data regions.
Views: Overlapping

- Each view may have a different consistency behaviour
- Views interact to represent the same data element
- Conceptual client CX proxies operations
- CX provided for free by view model
- Can use different data sharing semantics for same region

✓ Great for heterogeneous environments!
• Extension to overlapping views.
• *Mapping client* implements a mapping function that translates view operations.
Experiment: DSM Matrix Multiply

- 1200 x 1200 matrix multiply
- Cluster 1: Itanium 4-way SMP
- Cluster 2: Itanium 4-way ccNUMA + six Itanium 2-way SMP
- Cluster 2 has 1000Mbit internal switch
- Cluster 1 and 2 are connected via 100Mbit link
- Three view configurations: traditional, two domain, multi-protocol two domain.
Experiment: View Configurations

Environment:

- Multi-cluster of 20 CPUs.
Experiment: View Configurations

Traditional/single domain:

- Strict consistency over all nodes.
Experiment: View Configurations

Two locality domains:

- Two views of strict consistency.
Experiment: View Configurations

Two locality domains, multi-protocol:

• Two views of strict consistency.
• Internal multi-reader/multi-writer (MRMW) views on each multi-processor.
Matrix Multiply Results

- Poor traditional performance due to false-sharing
- Two domain improves performance by reducing inter-cluster communication
- Two domain with protocol selection improves performance further by utilising ccNUMA/SMP resources
- Clients on same node communicate using view interface operations
- Separation of client and protocol pager
• Pagers communicate with remote clients just like other clients
View Architecture

• Protocol selection requires only change of view specification
• No change to clients necessary
View Architecture

- Pagers communicate with different views just like other clients
- Views encapsulate a group of data sharers
### How do Views work?

#### Views Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update request</td>
<td>requests updates for given region</td>
</tr>
<tr>
<td>update propagate</td>
<td>propagate updates for given region</td>
</tr>
<tr>
<td>protection request</td>
<td>request access for given region</td>
</tr>
<tr>
<td>protection propagate</td>
<td>indication of new region access</td>
</tr>
<tr>
<td>token request</td>
<td>request a synchronisation token</td>
</tr>
<tr>
<td>token response</td>
<td>receive a synchronisation token</td>
</tr>
<tr>
<td>view create</td>
<td>create a new view</td>
</tr>
<tr>
<td>view select</td>
<td>select a view for use</td>
</tr>
<tr>
<td>view unselect</td>
<td>release a view</td>
</tr>
</tbody>
</table>
Creating and Using Views

Combination of three methods:

1. direct application use,
2. middleware or library,
3. system and administrative domains.

Example application use

```c
view1 = view_create (0, base, end, Strict);
...
view_select (view1);
```
Creating and Using Views

Combination of three methods:

1. direct application use,
2. middleware or library,
3. system and administrative domains.

Example application use

```c
view1 = view_create (0, base, end, Strict);
  ...
view2 = view_create (view1, base, end, MRMW);
  ...
view_select (view2);
```
Conclusions

Summary:

- Resource utilisation and performance improvements
- Protocol inter-operability avoids new hybrid protocols

Future work:

- Other programming models such as MPI.
  - Already examined single-sided MPI.
- Programming model interoperability
- Views for bulk data transfer, check-pointing etc.
- Comprehensive benchmarking: SPLASH, NAS, etc.
- Views in a wide area, single-system-image environment.
Adapting Distributed Shared Memory Applications in Diverse Environments

University of New South Wales, Sydney, Australia
danielp@cse.unsw.edu.au

Questions?
Application Interoperability

- Different view pagers communicate using view interface operations
- Mechanism for visualisation, multi-model applications, ...

Diagram:

- Node 1
  - LRC pager
  - MPI pager
  - Client

- Red arrows: view interface operations
- Black arrows: MPI operations