Distributed Systems

30. Distributed Shared Memory

Paul Krzyzanowski
Rutgers University
Fall 2014
Distributed Shared Memory (DSM) Goal

Allow networked computers to share a region of virtual memory

*How do we do this?*
Take advantage of the MMU

• Page table entry for a page is valid if the page is held (cached) locally

• Attempt to access non-local page leads to a page fault

• Page fault handler
  – Invokes DSM protocol to handle fault
  – Fault handler brings page from remote node

• Operations are transparent to programmer
  – DSM looks like any other virtual memory system
Simplest design

- Each page of virtual address space exists on only one machine at a time

- On page fault
  - Consult central server to find which machine is currently holding the page: Directory

- Request the page from the current owner:
  - Current owner invalidates PTE
  - Sends page contents
  - Recipient allocates frame, reads page, sets PTE
  - Informs directory of new location
Problem

• Directory becomes a bottleneck
  – All page query requests must go to this server

• Solution
  – Distributed directory
  – Distribute among all processors
  – Each node responsible for portion of address space
  – Find responsible system:
    • contact node[ page_num mod num_processors ]
**Distributed Directory**

Directory can be a bottleneck ⇒ distribute it

<table>
<thead>
<tr>
<th>Page</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>P3</td>
</tr>
<tr>
<td>0004</td>
<td>P1</td>
</tr>
<tr>
<td>0008</td>
<td>P1</td>
</tr>
<tr>
<td>000C</td>
<td>P2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>P3</td>
</tr>
<tr>
<td>0005</td>
<td>P1</td>
</tr>
<tr>
<td>0009</td>
<td>P0</td>
</tr>
<tr>
<td>000D</td>
<td>P2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002</td>
<td>P3</td>
</tr>
<tr>
<td>0006</td>
<td>P1</td>
</tr>
<tr>
<td>000A</td>
<td>P0</td>
</tr>
<tr>
<td>000E</td>
<td>--</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0003</td>
<td>P3</td>
</tr>
<tr>
<td>0007</td>
<td>P1</td>
</tr>
<tr>
<td>000B</td>
<td>P2</td>
</tr>
<tr>
<td>000F</td>
<td>--</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Replication

Multiple readers – Single writer
- Directory stores a copyset

Read operation:
- If page not local
  - Acquire read-only copy of the page
  - Set access writes to read-only on any writeable copy on other nodes

Write operation:
- If page not local or no write permission
  - Revoke write permission from other writable copies (if exists)
  - Get copy of page from owner (if needed)
  - Invalidate all copies of the page at other nodes
Improving performance

• Break rules to achieve better performance
  – But compiler and/or programmer should know what’s going on!

• Goals:
  – combat network latency
  – reduce number of network messages
Relaxing memory consistency

• Consider how we share memory in SMP systems
  – Work in critical sections
  – One process reading/writing
  – Nobody else accessing until process leaves critical section

• No need to propagate writes until process leaves its critical section
Release Consistency

• Separate synchronization into two stages:
  – 1. acquire access
    • Obtain valid copies of pages: check directory
  – 2. release access
    • Send invalidations for shared pages that were modified locally to the directory

```plaintext
acquire(R)  // start of critical section
Do stuff
release(R)  // end of critical section
```
Finer granularity

• Release consistency
  – Synchronizes all data
  – No relation between lock and data

• Use **object granularity** instead of **page granularity**
  – Each variable or group of variables can have a synchronization variable
  – Propagate only writes performed in those sections
  – Cannot rely on OS and MMU anymore
    • Need smart compilers
The End