Lazy Hybrid Release Consistency
Pulling better than pushing?
Mark Whitney

Overview
- Software DSM factoids
- Summary of RC implementations referenced
- Laziness?
- Update vs. Invalidate in a Lazy way
- Experiments they did

Software DSM tenets
- Communication overhead much higher than for custom MPs
- Sharing usually done at page granularity
  - Uses OS’s VM system to detect remote accesses
  - False sharing a big problem (big data objects)
- Communication bandwidth is more precious

Previous RCs

<table>
<thead>
<tr>
<th>Very Eager RC</th>
<th>Eager RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC al la DASH</td>
<td>Munin software DSM system</td>
</tr>
<tr>
<td>Writes pipelined within acquire-release pair</td>
<td>Remote access by OS VM</td>
</tr>
<tr>
<td>Invalidates immediately propagated</td>
<td>Writes buffered until release, on release, propagate to all sharers</td>
</tr>
<tr>
<td>Node only stalls if writes not complete at time of release</td>
<td>Could do invalidate or update</td>
</tr>
<tr>
<td>Allowed multiple writers</td>
<td></td>
</tr>
</tbody>
</table>
Previous RCs (cont.)

DASH

\[ P_1[x,y] \text{ acq } w(x) \text{ w(y) rel } \]

\[ P_2[x,y] \]

\[ P_3[y] \]

Munin

\[ P_1[x,y] \text{ acq } w(x) \text{ w(y) rel } \]

\[ P_2[x,y] \]

\[ P_3[y] \]

Lazy RC

- Assume all shared variables accesses are inside acquire/release pair
- Push relaxation a step further, only notify acquiring nodes
- No notifications of writes by release finish
  - Instead, send write notifications when next node acquires same lock
  - Notifications only sent to acquiring node

\[ P_1[x,y] \text{ acq } w(x) \text{ rel } \]

\[ P_2[x] \text{ acq } \text{ notify(x) rel } \]

\[ P_3[x] \text{ acq } \text{ notify(x) rel } \]

Release to Acquire on Lazy RC

- Upon release of a synch variable, node A does nothing
- Node B acquires synch variable
  - must locate node that released it
- Node B sends “vector timestamp” to node A for particular lock acquired
  - Timestamp identifies how long ago the shared state in acquire/release interval was updated
- Node B receives invalidates or “diff”ed updates from other nodes

Hybridization of Lazy RC

Problems with non-hybrids:

- Lazy invalidate creates more access misses
  - Shared data on by acquiring node is likely to be needed again
- Lazy update can be expensive
  - Acquiring node may have to communicate with many other nodes to get updated versions
- Lazy hybrid propagates invalidates unless last releasing node has most recent version, then updates
### Hybridization of Lazy RC (cont.)

**Lazy Invalidate**
- $P_1[x,y] \xrightarrow{\text{acq } w(x) \text{ rel}} P_1[x,y]$
- $P_2[y] \xrightarrow{\text{acq } w(y) \text{ rel}} P_2[y]$
- $P_3[x,y] \xrightarrow{\text{acq } r(y)} P_3[x,y]$

**Lazy Update**
- $P_1[x,y] \xrightarrow{\text{acq } w(x) \text{ rel}} P_1[x,y]$
- $P_2[y] \xrightarrow{\text{acq } w(y) \text{ rel}} P_2[y]$
- $P_3[x,y] \xrightarrow{\text{acq } r(y)} P_3[x,y]$

**Lazy Hybrid**
- $P_1[x,y] \xrightarrow{\text{acq } w(x) \text{ rel}} P_1[x,y]$
- $P_2[y] \xrightarrow{\text{acq } w(y) \text{ rel}} P_2[y]$
- $P_3[x,y] \xrightarrow{\text{inv}(x), \text{upd}(y)} P_3[x,y]$

### Simulation Details
- Execution-based simulation of 40MHz processor (up to 16 nodes)
- Ethernet modeled as broadcast bus, ATM as X-bar
- Simulated message time includes:
  - Base network latency
  - Additional delays from contention
  - Software overhead (on the order of 1000s of cycles)
  - Consistency state not included in message size, which may help lazy schemes since more state is needed

### The Experiments
- **Applications:**
  - Coarse grained sharing: TSP, Jacobi
  - Medium grained sharing: Water
  - Fine grained sharing: Cholesky
- Run some quick tests varying the network
  - ATM vs. ethernet
- Measure speedup and message traffic generated for each benchmark
- Look at the effect of software overhead, page size, proc speed, etc. on speedups

### Results
- First simple network experiment shows ATM does better
  - Not clear if this is due to b/w or contention
- Jacobi and TSP do not speed up much for eager vs. lazy policies
- Poster-boy Water speeds up a bunch with lazy, hybrid being best
  - Message counts and sizes alot less for lazy
- Cholesky scales badly for everything, slightly less bad for lazy
## Results II

- Higher b/w performs better
- ATM performs better than ethernet even with the same b/w due to contention
- Lower software overhead gives better performance, esp. in Water and Cholesky (finer grained)
- Speedup decrease for higher proc speeds show communication cost is more than just overhead
- Page size does not hurt lazy hybrid (others?)

## Conclusion

- Lazy release consistency works pretty well for “medium-grained” application (water)
- Not much room for improvement in coarse-grained, too much synchronization in fine-grained
- Good at reducing message traffic
- Clever idea but quite a bit more complicated than DASH RC
  - How much more state?