Transparently Consistent Asynchronous Shared Memory

Hakan Akkan, Latchesar Ionkov, & Michael Lang

LA-UR-13-20182
Motivation: Current Trends

- High core count sockets and nodes
- Power is a big concern
- Future architectures suggest
  - Lower clock speeds
  - Less memory per core
- New memory technologies
  - NVRAM
  - Memory Cube
- Therefore, intra-node sharing is becoming more important
Motivation: What problem are we trying to solve?

Data Movement/Memory requirements

- Per core memory is decreasing but total memory will be increasing
- There is more data to move around for analysis, visualization etc.
- New trend: Do it all in the node, while the application is running
  - Reduce the amount of data to store
  - Reduce the power/time to transmit data
- Applications share the node and the data is now local to two or more applications
Goals

TCASM – Shared memory for coupling independent applications

- **T** -- **Transparent**
  - No large code modifications
  - Publish version
  - Want to allow app to make progress

- **C** -- **Consistent**
  - Data doesn’t change while being processed

- **A** -- **Asynchronous**
  - Want to allow apps to make independent progress

- **SM** – **Shared Memory**.
Motivation: Who would benefit

- **Analytics**: reduce the data in the node
- **Visualization**: view the data in the node in REALTIME
- **Checkpointing**:
  - Burst-buffer – hierarchical storage, faster than PFS, close to node
- **NVRAM Management**
- **Debugging**: unobtrusively monitor what is happening inside the application
- **Application developers**
  - Just define data
  - No need to link processes
  - Application remain independent
Background/Related work

- **Checkpoint**: BLCR, CRIU
  - No for sharing between apps

- **Distributed memory**: MUNIN,

- **KSM**: Kernel same page merging
  - Kernel thread tries to merge pages with the same data
  - Also uses COW
  - MADVISE marks regions for merging
  - Single process space
  - No versioning

- **Virtual processes**: DUNE
  - Uses virtual processes (rather than virtual machines)
  - Can share pages, need common process parent
Solution

- **Producer / Observer(s) Model**
  - Producer publishes data at its natural interval
  - Consumer consumes at natural interval

- **Need a way for applications to share data with simple semantics**
  - Use existing MMAP system call

- **No synchronization**
  - Producer is never blocked.

- **Observer sees consistent view of data**

- **COW**
  - No wasted memory
  - Data is only copied when required
  - Memory use depends on how much memory producer modifies at each iteration
Current methods

- Single Copy – requires locking between processes, standard shared memory
  
  ![Diagram showing single copy method with producer, observer, lock, current, and old]

- Double-buffer – Two buffers, still need to coordinate the buffer swap
  
  ![Diagram showing double-buffer method with producer, observer, current, old, lock]

- Multibuffer – no synchronization, lots of memory (N copies, N-1 observers)
  
  ![Diagram showing multibuffer method with producer, current, old, older, oldest, lock, observer]
Implementation 1 – anon-asm

- New flag added to msync() call
- Uses a combination of “mmap”ed anonymous and file backed pages to manage versions of data
- Write protect pages to get notification when changed
  - Not cheap but already in use with many incremental checkpoint systems.
- This implementation has higher overhead with multiple observers

Walk through an example… next
Anon-asn

Producer

Page A

File Content

Page A

Observer

Page A

File-backed Page

Anonymous Page
Producer writes

<table>
<thead>
<tr>
<th>Producer</th>
<th>File Content</th>
<th>Observer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page A</td>
<td>Page A</td>
<td>Page A</td>
</tr>
</tbody>
</table>

File-backed Page
 Anonymous Page
Causes COW

Producer

File Content

Observer

COW

File-backed Page

Anonymous Page
Producer calls msync()

Producer: Page B  
File Content: Page A  
Observer: Page A

- File-backed Page
- Anonymous Page
Producer calls msync()
Producer writes again

Producer: Page B
File Content: Page B
Observer: Page A

File-backed Page
Anonymous Page
Causes COW

Producer

Page B

WRITE

File Content

Page B

Observer

Page A

COW

File-backed Page

Anonymous Page
Anonymous pages

- Producer: Page C
- File Content: Page B
- Observer: Page A

File-backed Page
Anonymous Page
Observers call `mmap()` for new version

- **Producer**: Page C
- **File Content**: Page B
- **Observer**: Page A
  - `mmap`

- **File-backed Page**
- **Anonymous Page**
Observers gets new version

Producer: Page C
File Content: Page B
Observer: Page B
File-backed Page
Anonymous Page
Implementation 2 – file-asm

- Again we use flags to msync
- When producers write we create a file-backed page, “unpublished”
- Then when producers call msync we swap files
  - Unlink old file, give unpublished file proper name
- Observer unmap – open “file” -- mmap to access latest copy
- Easier implementation
- Better performance with multiple observers, but there is a downside...
  - “file-asm” implementation incurs the "full-copy" penalty due to a Linux restriction that only allows pages to belong to a single file
  - A kernel module can remedy this, work-in-progress
Same flow, exchange of *named* files and *unpublished* files
Results – varying pages touched

```
Completion time (s) vs. Number of pages for different coverage and buffering settings:
- 100% coverage anon-asm
- 80% coverage anon-asm
- 100% coverage file-asm
- Double buffering
- 40% coverage anon-asm
- 20% coverage anon-asm

The graph shows a linear relationship between completion time and number of pages, with different lines representing different coverage levels and buffering strategies.

Notably, the 20% anon-asm setting has the steepest slope, indicating the highest completion time for a given number of pages.
```
Results 100% coverage, with 1 or 10 observers

![Graph showing completion time vs. number of pages for different scenarios, including Double buffering and 100% coverage file-asm with 1 or 10 observers.](image-url)
Application testing (new work since publication *)

- **Currently integrated with LANL mini-app SNAP (SN transport proxy)**
  - Implemented FORTRAN callable library in C
    - MMAP and MSYNC
  - Implemented consumer to copy data to stable storage (ie burst-buffer)
  - Tested with multiple threads

*Doug Otstott (Florida International University)*
“New” initial DATA

SNAP working set

% Pages changed

Iterations per MSYNC

0 20 40 60 80 100 120
1 2 4 10 25 50 75 100

SNAP working set
Conclusions

- Prototype shows benefit of approach
- COW shared memory for symbiotic applications
- No synchronization requirement
- Saves memory (SNAP use case)

Future Work

- Complete burst-buffer use case
- Additional use case for visualization (paraview)
- Performance with many producers/consumers in the system
- Looking at porting to Kitten/Palacious
Questions ???

Targeted for open source, paperwork in the system.

Thanks