Where Have all the Cycles Gone?

Investigating the Runtime Overheads of OS-Assisted Replication

Björn Döbel, Hermann Härtig
TU Dresden Operating Systems Group

Koblenz, 16.09.2013
Where have all the cycles gone?

ASTEROID – OS-Assisted Replication

Driver
Enc. Proc.
APP

L4Re
Romain

Fiasco.OC

ASTEROID – OS-Assisted Replication

- Interpose on system calls & CPU exceptions
- Replicate memory (no need for ECC)
- Unmodified binary applications
How Much is Replicated Execution?

- **Resource Overhead**
  - Roughly: $N \times$ replication $\rightarrow N \times$ resources
  - Optimizations vs. error coverage

- **Fault Coverage**
  - No complete measurements yet
  - Estimation: matches compiler-assistance

- **Runtime overhead**
  - Should be optimized for
  - This paper: SPEC INT 2006
Experiment Setup

Intel X5650 @ 2.66 GHz

L1

L2
Experiment Setup

12 GB RAM
ASTEROID – OS-Assisted Replication

- L4/Fiasco.OC, 32 bit + Romain
- SPEC INT 2006
  - 400.perl
  - 401.bzip2
  - 403.gcc
  - 429.mcf
  - 445.gobmk
  - 456.hmmer
  - 458.sjeng
  - 462.libquantum
  - 464.h264
  - 471.omnet++
  - 473.astar
  - 478.xalancbmk

12 GB RAM
400.perl

Normalized runtime overhead vs. native execution

Native
Single
DMR
TMR

0.879x
0.878x
0.876x
1.00x
The Problem: CPU Assignment

CPU0

App

print()

Higher priority logger

App

Native Execution

Time
The Problem: CPU Assignment

CPU0
- App
- Higher priority logger

CPU1
- App

CPU2
- App

CPU3
- App

Native Execution

Replicated Execution

print()
Where does overhead come from?

Sync time

Notification time
Where does overhead come from?

<table>
<thead>
<tr>
<th>Source</th>
<th>Overhead vs. native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per-replica execution</td>
<td>+/- 0</td>
</tr>
<tr>
<td>Sync time</td>
<td>~ 0</td>
</tr>
<tr>
<td>State comparison</td>
<td>~ 100 cycles</td>
</tr>
<tr>
<td>System call</td>
<td>~ 0</td>
</tr>
<tr>
<td>Notifications</td>
<td></td>
</tr>
<tr>
<td>Local core</td>
<td>~ 2,000 cycles</td>
</tr>
<tr>
<td>On socket</td>
<td>~ 6,000 cycles</td>
</tr>
<tr>
<td>Cross-socket</td>
<td>~ 14,300 cycles</td>
</tr>
</tbody>
</table>

Rule: Prefer placing replicas on the same CPU socket
Replicating SPEC INT 2006
Idea: Reduce Memory Management Overhead

- Assumption: memory management is expensive
- Idea: reduce overhead by using x86 huge pages (4 MB)
- Works for microbenchmark
- SPEC CPU: (nearly) no difference

Microbenchmark

4 KB pages

<table>
<thead>
<tr>
<th></th>
<th>Native: 0.72 s</th>
<th>1x: 0.80 s</th>
<th>2x: 2.23 s</th>
<th>3x: 3.12 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 MB pages</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Native: 0.38 s</td>
<td>1x: 0.38 s</td>
<td>2x: 0.53 s</td>
<td>3x: 0.91 s</td>
<td></td>
</tr>
</tbody>
</table>
Secondary Effects: Cache Miss Rates

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>DMR misses</th>
<th></th>
<th>TMR misses</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L2</td>
<td>L3</td>
<td>L2</td>
<td>L3</td>
</tr>
<tr>
<td>429.mcf</td>
<td>2,600</td>
<td>1,300,000</td>
<td>11,000,000</td>
<td>5,200,000</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>2,500</td>
<td>570</td>
<td>440,000</td>
<td>387,000</td>
</tr>
<tr>
<td>471.omnet++</td>
<td>270,000</td>
<td>6,900,000</td>
<td>35,000,000</td>
<td>21,200,000</td>
</tr>
</tbody>
</table>

Rule: Prefer placing replicas on a different CPU socket
### Secondary Effects: Cache Miss Rates

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>DMR misses</th>
<th>TMR misses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L2</td>
<td>L3</td>
</tr>
<tr>
<td>429.mcf</td>
<td>2,600 → 2,600</td>
<td>1,300,000 → 930,000</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>2,500 → 2,500</td>
<td>570 → 323</td>
</tr>
<tr>
<td>471.omnet++</td>
<td>270,000 → 290,000</td>
<td>6,900,000 → 5,500,000</td>
</tr>
</tbody>
</table>
SPEC INT: Improved L3 Miss Rates
And now for something slightly different...

Protecting the RCB

- We have full RCB source, so we can apply compiler-level techniques.

- Encoded compiler anyone? → Approximate Overhead
Protecting the RCB: 2nd try

Application Code

Kernel: System Calls

Romain Master Code

Additional Kernel Invocations

Hardware Stalls (e.g., caching)

Native execution time

Replication overhead

\[ t_{prot} := t_{app} + C \times (t_{kern} + t_{master} + t'_{kern}) + t_{hw} \]

\[ t_{hw} = t_{kern} = t'_{kern} = 0 \]

\[ t_{prot} := t_{native} + C \times t_{Replicated} \]
Protecting the RCB: 2nd try

- Selected values for C [3]:
  - $C_{SWIFT} = 1.09$
  - $C_{ANBD} = 3.89$

Summary

- Romain: <5% overhead for 3x replicating most of the SPEC INT 2006 benchmarks

- Replica-core placement matters

- RCB protection may add additional overheads
  - Combination with compiler-level methods seems feasible