Truly Non-blocking Writes

Luis Useche\textsuperscript{2}  Ricardo Koller\textsuperscript{2}  Raju Rangaswami\textsuperscript{2}  Akshat Verma\textsuperscript{1}

\textsuperscript{1}IBM Research, India

\textsuperscript{2}School of Computing and Information Sciences
College of Engineering and Computing

HotStorage Workshop, 2011
Memory access granularity is smaller than disk’s
Memory access granularity is smaller than disk’s ⇒ Writes to an out-of-core page require a full page fetch.
Memory access granularity is smaller than disk’s ⇒ Writes to an out-of-core page require a full page fetch.
Memory access granularity is smaller than disk’s $\Rightarrow$ Writes to an out-of-core page require a full page fetch.
Memory access granularity is smaller than disk’s ⇒ Writes to an out-of-core page require a full page fetch.
Memory access granularity is smaller than disk’s ⇒ Writes to an out-of-core page require a full page fetch.
Memory access granularity is smaller than disk’s \(\Rightarrow\) Writes to an out-of-core page require a full page fetch.
Memory access granularity is smaller than disk’s \(\Rightarrow\) Writes to an out-of-core page require a full page fetch.
Memory access granularity is smaller than disk’s ⇒ Writes to an out-of-core page require a full page fetch.

For writes: why wait for data that the application doesn’t need?
Memory access granularity is smaller than disk’s ⇒ Writes to an out-of-core page require a full page fetch.

For writes: why wait for data that the application doesn’t need?
Non-blocking Writes: Basic Approach

Process

OS

Backing Store
Non-blocking Writes: Basic Approach

1. Write($X$)
Non-blocking Writes: Basic Approach

1. Write($X$) → ??? → Miss → OS

Process → Backing Store
Non-blocking Writes: Basic Approach

1. Write(X) → ???
2. Miss
3. Buffer

Process → Backing Store → OS
Non-blocking Writes: Basic Approach

1. Write($X$)
2. Miss
3. Buffer
4. Issue

Process → ??? → OS

Patch

Backing Store
Non-blocking Writes: Basic Approach

1. Write($x$) → ??? → 2. Miss → ??? → 3. Buffer → Patch

4. Issue

5. Return

Process → Backing Store → OS
Non-blocking Writes: Basic Approach

1. Write($X$)
2. Miss
3. Buffer
4. Issue
5. Return
6. Complete

OS

Patch

Process

Backing Store

110
101
001
Non-blocking Writes: Basic Approach

1. Write\( (X) \)
2. Miss
3. Buffer
4. Issue
5. Return
6. Complete
7. Merge
Non-blocking Writes: Basic Approach

Benefits
1. Application execution time reduction
2. Increased backing store bandwidth usage
Motivation → Higher Fault Rates

Memory over-committed in virtualized environments

Source: VMware customer survey
Motivation → Higher Fault Rates

Memory over-committed in virtualized environments

More process running with multi-core and virtualized environments
Motivation → Higher Fault Rates

Memory over-committed in virtualized environments

More process running with multi-core and virtualized environments

Memory hierarchy moving towards a more active and faster backing store
We calculate the % of faults that can benefit in all our workloads
Motivation → % Non-blocking faults

- We calculate the % of faults that can benefit in all our workloads:

  - **Image Processing**  Rendering of SVG images
  - **Developer**  Unit and performance testing
  - **Server**  Application, database, and mail server
Motivation → % Non-blocking faults

- We calculate the % of faults that can benefit in all our workloads:
  - Image Processing  Rendering of SVG images
  - Developer  Unit and performance testing
  - Server  Application, database, and mail server
- Simulator with full-system memory traces.
- RAM set to 50% of app footprint
Motivation → % Non-blocking faults

- We calculate the % of faults that can benefit in all our workloads:
  - **Image Processing**  Rendering of SVG images
  - **Developer**  Unit and performance testing
  - **Server**  Application, database, and mail server
- Simulator with full-system memory traces.
- RAM set to 50% of app footprint
- Up to 80% of page faults benefit

![Bar chart showing % Non-Block Faults for different workloads: Image Proc, Developer, Server. Image Proc has the least benefit, Developer has a moderate benefit, and Server has the highest benefit.](chart.png)
Related Work

Alternatives to non-blocking writes:

**Perfect DRAM Provision**
Unpredictable or unbounded.

**Prefetching**
Can incur false positives and false negatives.

**Asynchronous System Calls**
1. Do not work with memory mapped pages
2. Written data *not* immediately available for reading
Solution Challenges

Process
Solution Challenges

```
write(buf, nbytes, dest_addr)
```

Diagram:
- Process
  - OS call
    - Store Inst
    - fault(dest_addr)
Solution Challenges

```
write(buf, nbytes, dest_addr)
patch{new_buf, nbytes, dest_addr}
fault(dest_addr)
```

Diagram:
- **Process**
  - OS call: write(buf, nbytes, dest_addr)
  - Store Inst: fault(dest_addr)
  - patch{new_buf, nbytes, dest_addr}
Solution Challenges

write(buf, nbytes, dest_addr)

patch{new_buf, nbytes, dest_addr}

fault(dest_addr)

Information Per Non-blocking Write

<table>
<thead>
<tr>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Offset</td>
</tr>
<tr>
<td>Data Written</td>
</tr>
<tr>
<td>Size of Data</td>
</tr>
</tbody>
</table>
Solution Challenges

**Process**

\[ \text{write(buf, nbytes, dest_addr)} \]

\[ \text{patch\{new_buf, nbytes, dest_addr\}} \]

**Information Per Non-blocking Write**

<table>
<thead>
<tr>
<th>Information</th>
<th>Supervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>write()</td>
<td>✔</td>
</tr>
<tr>
<td>Write Offset</td>
<td>✔</td>
</tr>
<tr>
<td>Data Written</td>
<td>✔</td>
</tr>
<tr>
<td>Size of Data</td>
<td>✔</td>
</tr>
</tbody>
</table>
Solution Challenges

- write(buf, nbytes, dest_addr)
- patch{new_buf, nbytes, dest_addr}
- fault(dest_addr)

Information Per Non-blocking Write

<table>
<thead>
<tr>
<th>Information</th>
<th>Supervised</th>
<th>Unsupervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Offset</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Data Written</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Size of Data</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Solution Challenges

write(buf, nbytes, dest_addr)

OS call

Store Inst

fault(dest_addr)

patch{new_buf, nbytes, dest_addr}

Information Per Non-blocking Write

<table>
<thead>
<tr>
<th>Information</th>
<th>Supervised</th>
<th>Unsupervised</th>
</tr>
</thead>
<tbody>
<tr>
<td>write()</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Write Offset</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Data Written</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Size of Data</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Fault</td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>
### Handling Unsupervised Writes

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Fast</th>
<th>All Arch?</th>
<th>Low Mem?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Feature Hard-ware</td>
<td>fault()</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
</tbody>
</table>

Diagram: 
- fault()
## Handling Unsupervised Writes

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Fast</th>
<th>All Arch?</th>
<th>Low Mem?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Feature Hardware</td>
<td>fault()</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opcode Disassembly</td>
<td>sw $t1, 0xff</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
sw $t1, 0xff
```

4 bytes offset data
## Handling Unsupervised Writes

<table>
<thead>
<tr>
<th>Approach</th>
<th>Description</th>
<th>Fast</th>
<th>All Arch?</th>
<th>Low Mem?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Feature Hardware</td>
<td>fault()</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Opcode Disassembly</td>
<td>sw $t1, 0xff data</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Page Diff-Merge</td>
<td>Disk Page or 0-buffer and 1-buffer</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Quantifying Benefits

1. Fraction of non-blocking write faults ✔
2. Outstanding write faults (over time)
3. Savings in execution time (new!)

Virtual Memory Simulator

<table>
<thead>
<tr>
<th>Input</th>
<th>RAM size &amp; Full System Memory Traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Performance statistics</td>
</tr>
</tbody>
</table>

- Memory size set to 50% of workloads footprint
- Creating patches is not required
How to measure the additional parallelism?

**Outstanding Write Faults (OWF):** # of parallel write faults at any time

- OWF \( \leq \) OIO
- OWF \( \leq \) 1 for single threaded applications
- OWF \( \geq \) 0 when using non-blocking writes

We need the variations over time as well

**\( E[OWF] \):** time-weighted average OWF
Quantifying Benefits → Time Reduction

- These results are not in the paper
- Execution time = Trace time + Synchronous read time
- Write time of dirty page on evictions ignored
- Rough estimate: error proportional to the number of dirty pages evicted
Conclusions and Future Work

- We presented non-blocking writes: a technique to eliminate read-before-writes
  - ✔ Reduced execution time
  - ✔ Increased device usage

- We estimate a reduction times of 0.1-54%

- In the future, we are planning to implement non-blocking writes to better study its implications
  - ✔ What workloads benefit from Non-blocking writes?
Questions?
Virtual Memory Simulator

**Input:** RAM size & Mem Traces

**Output:** Per Entry: Timestamp and event (hit, miss, evict);
Global: Performance stats.

- Writes to out-of-core pages considered non-blocking
- Non-blocking status revoked when:
  1. The page is read before I/O completion
  2. The page is evicted before I/O completion
Modified x86 software-MMU QEMU to log all memory accesses:
  - Instruction count, CR3, virtual/physical address, access-mode, page privileges.

### Workloads

<table>
<thead>
<tr>
<th>Type</th>
<th>#</th>
<th>Footprint Avg/Std (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>10</td>
<td>294/158</td>
</tr>
<tr>
<td>Developer</td>
<td>4</td>
<td>269/183</td>
</tr>
<tr>
<td>Image</td>
<td>1</td>
<td>149/0</td>
</tr>
</tbody>
</table>
1. Write in two pages: 0-page and 1-page.
2. Merge with and and or.

Process

Backing Store
1. Write in two pages: 0-page and 1-page.
2. Merge with *and* and *or*.

Process \[ \xrightarrow{1. \text{Write} \ 01} \]

Backing Store
1. Write in two pages: 0-page and 1-page.
2. Merge with *and* and *or*.

Process → 101

Back Up Store
1. Write in two pages: 0-page and 1-page.
2. Merge with *and* and *or*.

![Diagram showing the process of page diff-merge]

**Backing Store**
1. Write in two pages: 0-page and 1-page.
2. Merge with and and or.
1. Write in two pages: 0-page and 1-page.
2. Merge with *and* and *or*.
1. Write in two pages: 0-page and 1-page.
2. Merge with \textit{and} and \textit{or}.

\begin{itemize}
\item Process
\begin{itemize}
\item 1. Write
\item 2. Write
\end{itemize}
\end{itemize}

\begin{itemize}
\item Backing Store
\begin{itemize}
\item 3. Complete
\item And
\item Or
\end{itemize}
\end{itemize}