Non-Blocking Writes to Files

Daniel Campello, Hector Lopez, Luis Useche\textsuperscript{1}, Ricardo Koller\textsuperscript{2}, and Raju Rangaswami

\textsuperscript{1}Google, Inc. \quad \textsuperscript{2}IBM TJ Watson
Applications have different alternatives to persist data

Synchrony vs Asynchrony

- Synchronous Writes
  - Process
  - Page Cache
  - Backend Storage

- Asynchronous Writes
  - Process
  - Kernel Thread
  - Page Cache
  - Backend Storage
Memory access granularity is smaller than disk’s
⇒ Partial writes to pages not present in the cache require page fetch.

Why wait for data that the application doesn’t need?
Motivation: Breakdown of write operations

- On an average, **63.12%** of the writes involved partial page overwrites.
- Depending of page cache size, these overwrites could result in varying degrees of page fetches.
Motivation: LRU Cache Simulation

![Graph showing Partial Over-Writes to Out-of-core Page (%) vs. Page cache size (MB)]

- **ug-filesrv**
- **gsf-filesrv**
- **moodle**
- **backup**
- **usr1**
- **usr2**
- **facebook**
- **twitter**

Non-blocking Writes to Files
Non-blocking Writes: Asynchronous Fetch

**Benefits:**
1. Application execution time reduction
2. Increased backing store bandwidth usage
Non-blocking Writes: Asynchronous Fetch

Why issue a page fetch for data that the application doesn’t need?
Non-blocking Writes: Lazy Fetch

Benefits and Drawbacks:
1. Allocating page memory and fetching only if necessary
2. Resource utilization is unpredictable and can be bursty
Managing Page Fetches

Page Fetch Policies
- Asynchronous
- Lazy

Page Fetch Mechanisms
- Foreground
- Background
Non-blocking Writes to Files

Foreground vs Background Mechanisms

Metadata misses generate extra blocking fetches

Diagram showing the difference between foreground and background mechanisms.

- **Foreground**
  - Process
  - Page Cache
  - Backend Storage

- **Background**
  - Process
  - NBW Thread
  - Page Cache
  - Backend Storage
We found this problem in other OSes (FreeBSD, Xen)
We implemented non-blocking writes for files in the Linux kernel by modifying the generic virtual file system (VFS) layer

**Workloads**
- Filebench micro-benchmark
- SPECsfs2008 benchmark
- Mobibench system call trace-replay
• NBW-Async-FG: 50-60% performance improvement for Random Writes
Non-blocking Writes to Files

Filebench evaluation: Writes and Read-Writes

- **NBW-Async-FG**: 50-60% performance improvement for Random Writes
- **NBW-Async-BG**: 6.7x-30x (Random Writes) and 3.4x-20x (Random RW)
- **NBW-Lazy**: Up to 45x performance improvement
Filebench evaluation: Reads

Non-blocking writes do not affect the performance of read-only workloads
Non-blocking Writes to Files

**SPECsfs2008 evaluation**

- Around 70% write latency decrease
- NBW-Lazy read latency slightly affected by delayed fetching
- Overall latency is in direct relation to write and read latency
Mobibench System-call Trace Replay

Average operation latency decreased by 20-40%
Non-blocking Writes to Files

Related Work

- Huge body of work on optimizing writes performance
  - O_NONBLOCK flag for OPEN system call
    - If no data can be read or written, returns EAGAIN (or EWOULDBLOCK)
    - Requires application modification
  - Asynchronous I/O Library (POSIX AIO)
    - Multiple threads to perform I/O operations (expensive and scales poorly)
    - Requires application modification

Non-blocking writes do not require application modification
Conclusions

- Operating systems block processes on asynchronous writes in many cases!
- With non-blocking writes, we demonstrate that such blocking is unnecessary and detrimental to performance
- General solution for any kind of modern OS
- Does not require application modification
- Evaluation demonstrated how non-blocking writes effectively reduces write latency across a wide range of workloads
Thank You!

The traces used in this paper are available at
Non-blocking Writes to Files

Page Fetch Asynchrony

Blocking write

Non-blocking write

Waiting I/O: □

Thinking: □□
Page Fetch Parallelism

Blocking write

Write $P$  
Write $Q$  
Read $P$

Non-blocking write

Write $P$  
Write $Q$  
Read $P$

Blocking I/O:  
Background I/O:  

Non-blocking Writes to Files
## Motivation: Production workloads

### Production system trace descriptions

<table>
<thead>
<tr>
<th>Workload</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ug-filserv</td>
<td>Undergrad NFS/CIFS fileserver</td>
</tr>
<tr>
<td>gsf-filesrv</td>
<td>Grad/Staff/Faculty NFS/CIFS fileserver</td>
</tr>
<tr>
<td>moodle</td>
<td>Web &amp; DB server for department CMS</td>
</tr>
<tr>
<td>backup</td>
<td>Nightly backups of department servers</td>
</tr>
<tr>
<td>usr1</td>
<td>Researcher 1 desktop</td>
</tr>
<tr>
<td>usr2</td>
<td>Researcher 2 desktop</td>
</tr>
<tr>
<td>facebook</td>
<td>Mobibench facebook trace</td>
</tr>
<tr>
<td>twitter</td>
<td>Mobibench twitter trace</td>
</tr>
</tbody>
</table>
Highlights On Correctness

- Ordering of page updates
- Handling of disk errors
- Journaling File Systems
- OS-initiated page accesses
- Page persistence and syncs
- Handling read-write dependencies
- Multi-core and kernel preemption