Using Eager Strategies to Improve NFS I/O Performance

Stephen Rago, Aniruddha Bohra, Cristian Ungureanu
NEC Laboratories America
Introduction

• Background
  • Backup appliance development
  • NFS Version 3
  • Backup over NFS was slower than expected
    • With storage system capable of 400 MB/s, couldn’t saturate a 1Gb Ethernet
    • With 10Gb Ethernet, can’t approach throughput of storage subsystem
  • Built server testbed with conventional storage subsystem: ext3 on top of striped, 15K RPM disks
    • Server capable of 300 MB/s throughput to storage subsystem
NFS Performance Problems

- Streaming **write** performance erratic
  - Tuning the system to cache more data caused write throughput to vary from 40 MB/s to 200 MB/s on our test systems *for the same set of tunable values*
- Slow performance results from:
  - Multiple contexts writing generate out-of-order requests
  - Memory pressure leads to small, synchronous writes
  - Memory pressure also increases commits
- Streaming **read** performance lower than expected
  - Less than 100 MB/s on 10Gb Ethernet
  - Out-of-order requests defeat kernel read-ahead logic
Concurrency = Out-of-Order NFS Requests

<table>
<thead>
<tr>
<th></th>
<th>Reads</th>
<th>Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
<td>Read-ahead</td>
<td>Multiple writers (background flushing, pageout, and application) plus asynchronous writes</td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td>Multiple NFS threads</td>
<td>Multiple NFS threads</td>
</tr>
</tbody>
</table>
Problem 1: Synchronous Operations - Base System, Slow Run vs. Fast Run for Same Amount of Data Written

- **Asynchronous Writes**
  - Slow (40 MB/s): 56145
  - Fast (200 MB/s): 32855

- **Synchronous Writes**
  - Slow (40 MB/s): 207061
  - Fast (200 MB/s): 148

- **Commits**
  - Slow (40 MB/s): 34
Problem 2: Small Record Sizes - Idealized NFS Write Throughput
Problem 3: NFS Write Offset Ordering
(Writing a 32 GB File)
Problem 4: NFS Read Offset Ordering
(Reading a 32 GB File)
Solutions

• Three general techniques
  • Eager Writeback
    • Reduces concurrency on client and maintains sequentiality
  • Eager Page Laundering
    • Reduces client memory pressure
  • Request Ordering
    • Prevents out-of-order operations on a single file
• Implemented on Linux 2.6.36
• Techniques applicable to other operating systems
Technique 1: Eager Writeback

• Client-side mechanism
• Prevents application from creating dirty pages quickly
  • Pages written eagerly to server
  • Client waits for outstanding requests to complete before continuing

• Advantages
  • Starts sending dirty pages earlier -- better server utilization
  • Only one thread writes a file’s pages to the server
  • Better flow control

• Disadvantages
  • Starts sending dirty pages earlier -- limited page reuse for overwriting patterns
Simplified Page State Diagram

Free

Allocate

Active

Old/Modified

Inactive Dirty

Old/Unmodified

Writeback

Inactive Clean

Deallocate

Writeback

Unstable

Commit
Technique 2: Eager Page Laundering

- Client & Server mechanism
- Dirty pages on server eventually become clean
- Communicate *largest stable offset* from server to client
  - Piggybacked in NFS write response (takes half of verifier)
  - Negotiated at mount time
- Client reclaims (“launders”) pages eagerly
- Advantages
  - Reduces memory pressure on client
  - No commits or synchronous writes needed
- Disadvantages
  - Small protocol change
Largest Stable Offset (LSO)

Unmodified File Contents

Dirty  Dirty  Dirty

0

LSO
Largest Stable Offset (LSO)

Unmodified File Contents

Clean  Dirty  Dirty

0  LSO
Largest Stable Offset (LSO)

Unmodified File Contents

Clean

Dirty

Clean

↑

0

↑

LSO
Largest Stable Offset (LSO)

Unmodified File Contents

Clean

Clean

Clean

0

LSO

Tuesday, July 26, 2011
Base Client Page Counts

- Dirty Pages
- Unstable Pages
- Total
Base Client Page Counts

- Dirty Pages
- Unstable Pages
- Total

Number of Pages x 1000

Time (s)

Tuesday, July 26, 2011
Base Client Page Counts

- Dirty Pages
- Unstable Pages
- Total

Number of Pages x 1000

Time (s)
Base Client Page Counts

- Dirty Pages
- Unstable Pages
- Total

Number of Pages x 1000 vs. Time (s)

Total: 178 MB/s
Client Page Counts - Eager Writeback Only

- Dirty Pages
- Unstable Pages
- Total

Tuesday, July 26, 2011
Client Page Counts - Eager Writeback Only

Number of Pages x 1000

Dirty Pages
Unstable Pages
Total

Time (s)

Tuesday, July 26, 2011
Client Page Counts - Eager Writeback Only

- Dirty Pages
- Unstable Pages
- Total

Number of Pages x 1000

Time (s)

Tuesday, July 26, 2011
Client Page Counts - Eager Writeback Only

![Graph showing client page counts with time (s) on the x-axis and number of pages x 1000 on the y-axis. The graph compares Dirty Pages, Unstable Pages, and Total Pages.]

Tuesday, July 26, 2011
Client Page Counts - Eager Writeback Only

- Dirty Pages
- Unstable Pages
- Total

Number of Pages x 1000

Time (s)

230 MB/s
Client Page Counts - Eager Writeback & Eager Page Laundering

- Dirty Pages
- Unstable Pages
- Total
Client Page Counts - Eager Writeback & Eager Page Laundering

Number of Pages x 1000

- Dirty Pages
- Unstable Pages
- Total

Time (s)

Tuesday, July 26, 2011
Client Page Counts - Eager Writeback & Eager Page Laundering

- Dirty Pages
- Unstable Pages
- Total

Number of Pages x 1000

Time (s)

Tuesday, July 26, 2011
Client Page Counts - Eager Writeback & Eager Page Laundering

- Dirty Pages
- Unstable Pages
- Total

Number of Pages x 1000

Time (s)
Client Page Counts - Eager Writeback & Eager Page Laundering

Number of Pages x 1000

Dirty Pages
Unstable Pages
Total

Time (s)

244 MB/s

Tuesday, July 26, 2011
Technique 3: Request Ordering

- Server sorts requests based on RPC transmission ID
- Server-side mechanism
- Prevents out-of-order completion of requests from competing threads

**Advantages**
- Improves sequential read performance
- When used during writes, can further improve read performance (depending on file system implementation)

**Disadvantages**
- Adds a small delay (50 ns) on reads to facilitate sorting, but only for sequential reads on files where the queue is empty
Sorting Request on the Server

Head of Queue
Sorting Request on the Server
Sorting Request on the Server

Head of Queue  2  4
Sorting Request on the Server

Head of Queue: 1 2 4
Sorting Request on the Server

Head of Queue

1  2  4  6
Sorting Request on the Server

Head of Queue

1 2 4 5 6
Sorting Request on the Server

Head of Queue: 1 2 3 4 5 6
NFS Write Offset Ordering

- Base
- Eager
NFS Write Offset Ordering

![Diagram showing NFS Write Offset Ordering with red and blue lines representing Base and Eager, respectively.](image)
NFS Write Offset Ordering

Write Offset (GB)

Request Number

- Base
- Eager
NFS Read Offset Ordering

- Base
- Eager
NFS Read Offset Ordering

![NFS Read Offset Ordering Graph]

- Red: Base
- Blue: Eager

Read Offset (MB) vs. Request Number

Tuesday, July 26, 2011
NFS Read Offset Ordering

![Graph showing NFS Read Offset Ordering with two lines representing Base and Eager]

- Request Number
- Read Offset (MB)
Performance Comparisons

- Micro benchmarks
  - Streaming I/O
  - Random Writes
  - Non-sequential Writes
  - Adversarial Page Reuse
- Macro benchmarks
  - Filebench Fileserv
  - Filebench Videoserv
Streaming I/O Performance

Read

Write
Streaming I/O Performance

Throughput (MB/s)

- Read
- Write

Local

- Read: 326 MB/s
- Write: 279 MB/s

NFS Base

NFS Eager

Tuesday, July 26, 2011
Streaming I/O Performance

Throughput (MB/s)

- Local: Read 326, Write 279
- NFS Base: Read 96, Write 178
- NFS Eager: Read 96, Write 178
Streaming I/O Performance

- **Local**: 326 Read, 279 Write
- **NFS Base**: 96 Read, 178 Write
- **NFS Eager**: 321 Read, 244 Write
Random Write Performance

Throughput (MB/s)

Record Size

Base

Eager

Tuesday, July 26, 2011
Nonsequential Write Performance

Throughput (MB/s)

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Base</th>
<th>Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 KB Alternating</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>256 KB Alternating</td>
<td>194</td>
<td>235</td>
</tr>
<tr>
<td>1MB/11MB Strided</td>
<td>177</td>
<td>208</td>
</tr>
</tbody>
</table>

Tuesday, July 26, 2011
Adversarial Example

1.5 GB --
1.4 GB -- dirty_background_ratio

0 GB --
Adversarial Example

1.5 GB -- dirty_background_ratio
1.4 GB --

<table>
<thead>
<tr>
<th>Footprint</th>
<th>Base</th>
<th>Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 GB --
Adversarial Example

1.5 GB -- dirty_background_ratio -- 1.4 GB
0 GB --

<table>
<thead>
<tr>
<th>Footprint</th>
<th>Base</th>
<th>Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Adversarial Example

<table>
<thead>
<tr>
<th>Footprint</th>
<th>Base</th>
<th>Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5 GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4 GB</td>
<td>dirty_background_ratio</td>
<td></td>
</tr>
<tr>
<td>0 GB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Adversarial Example

1.5 GB -- dirty_background_ratio
1.4 GB --

<table>
<thead>
<tr>
<th>Footprint</th>
<th>Base</th>
<th>Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4 GB (32 GB total)</td>
<td>1093 MB/s (1.4 GB to disk)</td>
<td>513 MB/s (18 GB to disk)</td>
</tr>
</tbody>
</table>
Adversarial Example

![Diagram showing footprints and data transfer rates]

<table>
<thead>
<tr>
<th>Footprint</th>
<th>Base</th>
<th>Eager</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4 GB (32 GB total)</td>
<td>1093 MB/s (1.4 GB to disk)</td>
<td>513 MB/s (18 GB to disk)</td>
</tr>
<tr>
<td>1.5 GB (32 GB total)</td>
<td>469 MB/s (18 GB to disk)</td>
<td>253 MB/s (32 GB to disk)</td>
</tr>
</tbody>
</table>
Filebench Fileserver Workload

Throughput (Operations/sec) vs Number of Clients

- Base
- Eager

Tuesday, July 26, 2011
Filebench Videoserver Workload

- Throughput (Operations/sec)
- Number of Clients
- Base
- Eager
## Implementation

<table>
<thead>
<tr>
<th>Technique</th>
<th>Scope</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eager Writeback</td>
<td>NFS Client</td>
<td>100</td>
</tr>
<tr>
<td>Eager Page Laundering</td>
<td>NFS Client &amp; Server</td>
<td>150</td>
</tr>
<tr>
<td>Request Ordering</td>
<td>NFS Server</td>
<td>120</td>
</tr>
</tbody>
</table>
Related Work

  *Eager Writeback - A Technique for Improving Bandwidth Utilization* (33rd ACM/IEEE Symposium on Microarchitecture)
- Ellard & Seltzer 2003
  *NFS Tips and Benchmarking Traps* (USENIX ATC)
- Ellard, et al. 2003
  *Passive NFS Tracing of Email and Research Workloads* (FAST ’03)
  *CA-NFS: a Congestion-Aware Network File System* (FAST ’09)
Summary

• For writes, memory pressure leads to performance problems
• For reads, out-of-order requests disable read-ahead
• Eager writeback, eager page laundering, and request ordering improve sequential throughput
• No harm for many nonsequential workloads
  • May even improve throughput when clients experience memory pressure