Lustre Metadata Writeback Cache
Design and implementation

Yingjin Qian, Oleg Drokin
Why (Metadata) Write Back Caching for Lustre?

• Cache is the key for good performance
  • Page cache, inode cache, dentry cache
• Data is well cached in Lustre
  • Page cache for both data writing and reading
• No client cache for metadata changes
  • Each metadata modification sent to MDS
• Metadata performance is important
  • Applications create many files today
  • Millions of RPCs sent over network
Current Data Cache/Acceleration Inside Lustre

- **Persistent Client Cache**
  - Local storage on clients for read-only or exclusive files
- **Lustre on Demand to cache file sets of jobs**
  - Quicker client networks and storage for running jobs
- **Data on MDT for data acceleration**
  - Less RPC and quick MDT for small files
- **OST pool on SSD for cache**
  - Quicker OSTs for hot data
- **Data read/write are fully cached**
  - LDLM lock protects data consistency
  - Page level cache management
- **Metadata needs acceleration too!**
When Metadata is Nothing Special

- Shared block filesystems have it easy – metadata is just data
  - Client locks the block(s), reads and interprets the contents
  - Perfect cache for both reads and modifications, just like a local filesystem
  - Crumbles under contention as lock-read-modify-write-unlock cycles get expensive fast

- To address the contention various tricks are played
  - Various libraries embed subdirectory trees inside specially formatted files are common
  - Minimize roundtrips by trying to send updates directly between clients (GPFS)
  - Complicate matters by reducing lockable block size
When Metadata is Unique

- Lustre is not a shared block filesystem
  - Metadata is interpreted on the server
  - Client receives piecemeal bits, that allows each one to be locked/cached separately
  - Changes are sent piecemeal, no need to read entire directory to create a new name
  - This gets expensive when there is no contention

- To address uncontended cases some tricks are played
  - Block-based images of filesystems for “filesets” that are separately mounted (CCI)
When Metadata is Just the VFS

- Exclusive lock at the root of subtree
  - The subtree could be populated by new creates (common)
  - Or reading data from the server
- All the operations then become node local
  - It’s like a shared block filesystem without a block underneath
  - Super low latency
- Granularity of this lock is “whole subtree underneath”
General Idea of Lustre WBC

Lustre Client

Batched & Delayed & Aggregated Metadata Flush

MDS

Lustre Client

- Normal Lustre tree
- Tree cached locally on client
- Tree not yet flushed to MDS
Batching of Metadata Modifying Operations

- Now with full cache of modifications we can also batch updates
- One RPC brings along many changes to the server
- Some updates could be coalesced locally and even reduced to nothing
  - `touch file; chmod o-r file; mv file file2; rm file2` => no RPCs
- Some audit folks might not be happy about this though
Batching of Read Operations

- Lustre already has a “statahead” metadata readahead
- Makes a good natural first step to showcase batched RPC functionality
- Will plug into the batched RPC mechanism nicely
- Coming in Lustre 2.16
- Will aggregate getattr RPCs for statahead
- Detects breadth-first (BFS) or Depth-first (DFS)
  - Direct statahead to next file/subdirectory based on tree walk pattern
  - Detect strided pattern for alphanumeric ordered traversal + stat()
    - e.g. file00001, file001001, file002001… or file1, file17, file31,… order
  - IO500 mdtest-{easy/hard}-stat performance improved 77%/95%
Handling Contention in a Cached Directory Tree

- When second client tries to access files in the cached directory tree
- Bump into EXclusive lock at top level of the tree
- Lock holder is asked to release the lock
- Flushes top level of entries, obtaining EX locks on new subdirectories
- Another client can now see and descend into next subdirectory level
- Repeat as needed for second client to access subdirectory tree
- No need to flush entire subtree at once to have global visibility
Client 1 has a cached subdirectory tree starting from entry 'linux' with EXclusive lock on it (e.g. after fetch kernel).
Contestion and Global Visibility

Client 2 requests access to pathname in Client 1's cached subdirectory tree starting from topmost uncached entry 'linux', MDS passes request on to Client 1.
Client 1 creates MDS inodes for each entry in 'linux' directory and requests EXclusive lock on each one. 'linux' directory is entirely Flushed to MDS.
Contestation and Global Visibility

Client 2 next requests 'drivers' entry in subdirectory from MDS, passed on to Client 1. Other entries in 'linux' remain in Client 1 cache undisturbed.
Client 2

ls git/linux/drivers/net/Makefile

Client 1 again creates MDS inodes for each subdirectory entry and requests EX locks for them; 'drivers' is nowFlushed to MDS.

Contention and Global Visibility
Contention and Global Visibility

Client 2 and Client 1 repeat steps for last level of directory tree until 'Makefile' is Flushed and accessible to Client 2. Other parts of tree not flushed.
Main Usage Targets for Lustre WBC

• Client-side metadata writeback cache instead of server-side
  • Pros: higher acceleration allowed by metadata locality
  • Cons: more complex mechanisms to keep consistency
• Delayed and grouped metadata flush instead of immediate RPC to MDS
  • Pros: many fewer MDS interactions for better performance
  • Cons: mechanism needed for batched flush and space/inode reservation
• Cache in volatile memory instead of persistent storage
  • Pros: quickest storage type
  • Cons: need to flush frequently to reduce risk of data loss
• Keep strong POSIX semantics instead of loosening semantics
  • Pros: transparent and standard behavior for applications on multiple clients
  • Cons: complex LDLM lock protection to maintain consistency
Flushing and Memory Control

• Data and metadata flush happens when:
  • Access of the directory tree from remote clients
  • Memory pressure on local host
  • Periodic aging of cache

• Quick flush from client cache to MDTs
  • Metadata flushing will use bulk RPC for batched flush
  • Only flush or degrade part of the directory tree rather than entire tree
Components in Lustre WBC

- Call backs
- Cache Policy of WBC
- PCC interface of WBC
- LDLM
- MDC
- PCC on Local FS (future)
- Data & Metadata Flush
- LDLM Lock Reclaim

VFS

- Dentry Cache
- Inode Cache
- Page Cache

MDS/OSS
Assimilation of File Data in WBC

• WBC manages both cached metadata and data

• What is WBC-Assimilation of data?
  • Move page cache from being managed by WBC to being managed by Lustre client
  • Data is still in page cache of Lustre client, not flushed to OSS yet

• When to WBC-Assimilate data?
  • Before flushing data to OSS, a WBC-cached file need to be WBC-Assimilated

• How to WBC-Assimilate data?
  • Metadata of the file and its ancestors need to be flushed first
  • File layout created on MDT
  • Put all page cache of the file under the management of main Lustre
  • Now file data could be flushed to storage servers too
Features and Advantages of WBC

• WBC flushes metadata of file in batch  
  • > 1000 file updates in a single bulk RPC
• Batch operations of metadata can be used to delete a whole directory  
  • Accelerates “rm -rf” a lot
• WBC aggregates metadata updates within a short interval  
  • Only the final state of metadata will be flushed to MDS  
  • Multiple operations aggregated into a single RPC
• WBC can be integrated with Persistent Client Cache (PCC)  
  • Data will still be cached in PCC after WBC-Assimilation  
  • Keep more data local to client, more RAM for metadata caching
• Possible offline/disconnected operations on Lustre client in the future
Evaluation: Single Node Metadata Performance

IOPS (ops/sec)

<table>
<thead>
<tr>
<th></th>
<th>NFS</th>
<th>Lustre</th>
<th>MetaWBC</th>
<th>Ext4</th>
<th>tmpfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir creation</td>
<td>10^4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dir stat</td>
<td>10^5</td>
<td>10^7</td>
<td>10^7</td>
<td>10^6</td>
<td></td>
</tr>
<tr>
<td>dir removal</td>
<td>10^6</td>
<td>10^5</td>
<td>10^6</td>
<td>10^5</td>
<td>10^6</td>
</tr>
<tr>
<td>file creation</td>
<td>10^4</td>
<td></td>
<td>10^5</td>
<td>10^4</td>
<td>10^5</td>
</tr>
<tr>
<td>file stat</td>
<td>10^5</td>
<td>10^6</td>
<td>10^6</td>
<td>10^5</td>
<td>10^6</td>
</tr>
<tr>
<td>file removal</td>
<td>10^6</td>
<td>10^5</td>
<td>10^6</td>
<td>10^5</td>
<td>10^6</td>
</tr>
</tbody>
</table>

Single node mdtest metadata performance (16 processes @ 100K files and directories)
Evaluation: Multiple Node Metadata Performance

IOPS (ops/s)

MetaWBC mdtest metadata performance scaling

Number of Nodes

1 2 4 8 16
Evaluation: Real-world Workloads

• Compare various workloads to other file systems on a single node
  • filebench default workloads (1 minute runtime)
  • Common command line applications operating on the Linux kernel source code

<table>
<thead>
<tr>
<th>App.</th>
<th>NFS</th>
<th>Lustre</th>
<th>MetaWBC</th>
<th>Ext4</th>
<th>tmpfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp -rf</td>
<td>64.00</td>
<td>48.00</td>
<td>5.10</td>
<td>1.24</td>
<td>0.75</td>
</tr>
<tr>
<td>find -uid</td>
<td>2.89</td>
<td>2.48</td>
<td>0.10</td>
<td>0.71</td>
<td>0.09</td>
</tr>
<tr>
<td>du -s</td>
<td>2.84</td>
<td>2.34</td>
<td>0.10</td>
<td>0.70</td>
<td>0.08</td>
</tr>
<tr>
<td>ls -IRU</td>
<td>3.40</td>
<td>1.57</td>
<td>0.28</td>
<td>0.93</td>
<td>0.29</td>
</tr>
<tr>
<td>grep -r</td>
<td>15.59</td>
<td>16.58</td>
<td>1.08</td>
<td>15.82</td>
<td>0.46</td>
</tr>
</tbody>
</table>
Evaluation: Pathological Workload

• Investigate pathological workload for write-back caching
  • mpiFileUtils dtar: Parallel extraction of eight Linux Kernel source code trees
  • EX locks are granted when creating a directory or during de-rooting
  • EX locks immediately revoked due to conflicting access from remote clients
  • Constant flush-back of cached files and transition to write-through mode

<table>
<thead>
<tr>
<th>Time phase</th>
<th>Create tree</th>
<th>Extract data</th>
<th>Update attr</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>CephFS</td>
<td>87</td>
<td>180</td>
<td>59</td>
<td>326</td>
</tr>
<tr>
<td>Ceph_async</td>
<td>89</td>
<td>170</td>
<td>62</td>
<td>321</td>
</tr>
<tr>
<td>Lustre</td>
<td>13</td>
<td>76</td>
<td>28</td>
<td>197</td>
</tr>
<tr>
<td>MetaWBC</td>
<td>1</td>
<td>56</td>
<td>1</td>
<td>136</td>
</tr>
</tbody>
</table>

Time (in seconds) for mpiFileUtils dtar phases

Even under worst-case workloads for writeback caching, WBC improves Lustre metadata performance
Evaluation: Untar of WBC Against Other File Systems

Lustre: DDN Al400X Appliance (20 X SAMSUNG 3.84TB NVMe, 4X IB-HDR100)
Lustre clients: Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1
Local File System on SSD: Intel SSDSC2KB240G8

Time Cost of Decompressing Linux Kernel Source Code Tarball

<table>
<thead>
<tr>
<th>File System</th>
<th>tar (s)</th>
<th>tar.gz (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmpfs</td>
<td>0.7</td>
<td>4.8</td>
</tr>
<tr>
<td>Ext4</td>
<td>1.3</td>
<td>4.8</td>
</tr>
<tr>
<td>NFS</td>
<td>315</td>
<td>308</td>
</tr>
<tr>
<td>Lustre</td>
<td>82</td>
<td>81</td>
</tr>
<tr>
<td>Lustre on WBC</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Type</th>
<th>tar (s)</th>
<th>tar.gz (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tar.gz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Metadata Performance of WBC Against Network FS

Lustre: DDN AI400X Appliance (20 X SAMSUNG 3.84TB NVMe, 4X IB-HDR100)
Lustre clients: Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1
Local File System on SSD: Intel SSDSC2KB240G8
Metadata Performance of WBC Against Local FS

Lustre: DDN Al400X Appliance (20 X SAMSUNG 3.84TB NVMe, 4X IB-HDR100)
Lustre clients: Intel Gold 5218 processor, 96 GB DDR4 RAM, CentOS 8.1
Local File System on SSD: Intel SSDSC2KB240G8
Summary

• Metadata Writeback Cache will accelerate metadata intensive workloads
• Batched RPC support and improved statahead coming in Lustre 2.16
• Complete WBC feature targeted for Lustre 2.17
Thank you!