Taming Metadata Storms in Parallel Filesystems with MetaFS

Tim Shaffer
A (well-meaning) user tried to run a bioinformatics pipeline to analyze a batch of genomic data.
Shared filesystem performance became degraded, with other users unable to access the filesystem.
That user got a strongly worded email and had to stop their analyses.
Certain program behaviors produce large bursts of metadata I/O activity (e.g. library search).

These behaviors can occur at the same time across multiple workers (e.g. startup, new analysis phase).

With a large number of nodes, the timing and intensity of metadata activity align to overwhelm the shared FS.
Shared filesystems can scale up their metadata capacity.

Panasas, Ceph, etc. use multiple metadata servers to better distribute the load.

General purpose solution
Applications can use a metadata service layered on top of the shared filesystem (e.g. BatchFS, IndexFS).

More efficient metadata management than the native filesystem.

Allows for client-side caching and batch updates.
Changes to the filesystem interface that allow weaker consistency or bulk operations

statlite and getlongdir system calls are examples.

This approach is not widely implemented.
Spindle provides library loading as a service.

Hooks into the dynamic loader on each node and builds an overlay network.

Nodes load shared objects by contacting each other rather than reading from the shared FS every time.
Case Study: MAKER

MAKER is a bioinformatics pipeline for analyzing raw gene sequence data.

It builds an annotated genome database with information on sequence repeats, proteins, etc.

http://www.yandell-lab.org/software/maker.html
Case Study: MAKER

MAKER presents a number of challenges at scale

- Large number of software dependencies (OpenMPI, Perl 5, Python 2.7, RepeatMasker, BLAST, several Perl modules)
- Composed of many sub-programs written in different languages (Perl, Python, C/C++)
- Installation consists of 21,918 files in 1,757 directories
- Unusual metadata load on shared filesystems
- Prone to causing a metadata storm
To help identify the causes of MAKER’s performance issues, we used `strace` to record syscalls made during an analysis.

For each syscall, we captured the type, timestamp, and paths/file descriptors used.

We also `straced` all children to capture sub-programs.
Profiling MAKER’s I/O Behavior

18212 1503501245.079960 read(3;/lib64/libpthread-2.12.so>, "\x7f\x45\x4c\x46\x02\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03\x00\x3e\x00\x01\x00\x00\x00"..., 832) = 832
Grouped relevant syscalls as

- **data** *(read, readv, write, ...)*
- **metadata** *(stat, readdir, readlink, open, ...)*

and by location

- Working directory (CWD)
- `/tmp`
- Shared FS
- Local system (`/bin, /usr/...`)**
## I/O Activity by Filesystem Location

<table>
<thead>
<tr>
<th>Filesystem Location</th>
<th>Access Mode</th>
<th>I/O Ops</th>
<th>Bandwidth (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWD</td>
<td>RW</td>
<td>257,060</td>
<td>1,435,228,808</td>
</tr>
<tr>
<td>/tmp</td>
<td>RW</td>
<td>1,163,711</td>
<td>2,463,335,142</td>
</tr>
<tr>
<td>Shared FS</td>
<td>RO</td>
<td>1,512,545</td>
<td>2,807,495,139</td>
</tr>
<tr>
<td>Local System</td>
<td>RO</td>
<td>906,327</td>
<td>68,929,672</td>
</tr>
</tbody>
</table>
As suspected, MAKER causes large bursts of metadata activity.

Intermediate and output data contribute relatively little to metadata activity over the course of an analysis.

Largest contributor is subprogram startup/library loading.
Panasas ActiveStor 16 filesystem
- 7 Director Blades + 70 Storage Blades
- Up to 84 Gb/s read bandwidth
- Up to 94,000 IOPS while reading data

We used a synthetic benchmark (\texttt{ls -r} in a directory tree with 74,256 files and 4,368 directories) to measure pure metadata performance.
Running Times for Parallel Benchmark Instances

<table>
<thead>
<tr>
<th>Parallel Instances</th>
<th>Instance Running Time (s)</th>
<th>Total Metadata I/O Operations</th>
<th>Average FS MIOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.7</td>
<td>179,091</td>
<td>13,038</td>
</tr>
<tr>
<td>4</td>
<td>22.6</td>
<td>716,364</td>
<td>31,664</td>
</tr>
<tr>
<td>8</td>
<td>41.9</td>
<td>1,432,728</td>
<td>31,194</td>
</tr>
<tr>
<td>16</td>
<td>86.1</td>
<td>2,865,456</td>
<td>33,262</td>
</tr>
<tr>
<td>24</td>
<td>130.6</td>
<td>4,298,184</td>
<td>32,916</td>
</tr>
</tbody>
</table>
To reduce shared FS load, we considered

- Local installation
- Disk image
- Containers (Docker, Singularity, ...)
- Filesystem overlay

These depend on availability at the site.
Software installation does not change during an analysis.

We can index the software installation metadata.

- Trade **numerous metadata operations** for a **single file read**
- Library is search handled locally
We implemented MetaFS as a FUSE module for evaluating this approach.

- Transparent overlay applied to an existing directory
- Easy to add/remove without modifying your scientific app
- Reads metadata index at startup and presents a read-only view of the software installation
1. Directory search
Normal Access

2. Read data

```
/dir1
├── file1
└── dir2
    ├── file2
    └── file3
└── dir3
```

```
010010
110010
...
101101
010101
...
010101
101010
...
```
1. Read metadata

/scratch
├── dir1
│   └── file1
├── dir2
│   ├── file2
│   └── file3
└── dir3

010010
110010
...
101101
010101
...
010101
101010
...
Create Index

2. Write Index File

Index

```
/scratch
├── dir1
│   └── file1
├── dir2
│   ├── file2
│   └── file3
└── dir3
```

```
010010
110010
...
101101
010101
...
010101
101010
...
```
Using MetaFS

1. Read index (startup only)
Using MetaFS

2. Directory search

```
/scratch
├── dir1
│   └── file1
├── dir2
│   ├── file2
│   └── file3
└── dir3
```

```
010010 110010 ...
101101 010101 ...
010101 101010 ...
```

```
Index
```

```
W MetaFS
W MetaFS
W MetaFS
```
Using MetaFS

3. Read data

/index
├── dir1
│   └── file1
├── dir2
│   ├── file2
│   └── file3
└── dir3

MetaFS

W

W

W

010010
110010
...

010101
101010
...

011011
010101
...

Index
For the `ls` benchmark with MetaFS in place, running time was on par with single-instance performance regardless of the number of parallel instances.

We also ran MAKER with MetaFS in place over the software installation directory.

MAKER requires **no modification** to run with MetaFS.
When starting, MetaFS reads the index file (~2 MB for MAKER’s installation directory).

Metadata activity to the shared FS is **significantly reduced** at the cost of a **small increase in data transfer** (index file).

No observed performance decrease due to FUSE.
Reduction in Metadata Load on the Shared Filesystem with MetaFS

<table>
<thead>
<tr>
<th></th>
<th>Metadata Ops.</th>
<th>Data Transfer (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls</td>
<td>179,091</td>
<td>0</td>
</tr>
<tr>
<td>ls + MetaFS</td>
<td>8,738</td>
<td>4,900,655</td>
</tr>
<tr>
<td>MAKER</td>
<td>1,142,781</td>
<td>2,807,495,139</td>
</tr>
<tr>
<td>MAKER + MetaFS</td>
<td>14,726</td>
<td>2,809,472,114</td>
</tr>
</tbody>
</table>
Based on the number of I/O ops. and the measured capacity of the system, a single user would saturate the shared FS with an average of 66 instances of MAKER running in parallel.

Bursty activity could reduce this limit further.

With MetaFS in place, we can remove this limit, allowing an estimated 5,000 parallel instances (∗).
MetaFS significantly reduces the (often unnecessary) metadata I/O encountered during program startup.

Local indexing is a lightweight approach: no changes to application or infrastructure necessary.

A major challenge for users is identifying when to apply optimizations. This is easy for software installations.
Tim Shaffer

tshaffe1@nd.edu

github.com/trshaffer