Lustre – Finding the Filesystem Bottleneck

Daniel Kobras
science + computing ag
IT-Dienstleistungen und Software für anspruchsvolle Rechnernetze
Tübingen | München | Berlin | Düsseldorf
**science+computing**

| Founded in | 1989 |
| Offices | Tuebingen |
| | Munich |
| | Duesseldorf |
| | Berlin |
| Employees | 268 |
| Shareholder | Bull S.A. (100%) |
| Turnover 10/11 | 26.7 Mio. EUR |

**Portfolio**

**IT Service** for complex computing environments
Complete solutions for Linux- and Windows-based **HPC**

**scVENUS** System management software for efficient administration of homogeneous and heterogeneous environments
Motivation

'Dear admin, filesystem is slow, please fix'

- Performance problems are among the hardest problems to debug
- Often no error messages available
- Finding root cause is hard, especially in distributed systems comprising of many components

- May not be an actual problem at all, but
  - Overload from legitimate use
  - Overload because of (deliberate) imbalanced sizing
  - Unrealistic expectations
Performance debugging roadmap

• Check servers
  • Are the servers (over)loaded?
  • Which servers are (over)loaded?
  • Which operations are (over)loading the filesystem?
  • Which clients are (over)loading the filesystem?

• Check clients
  • Which processes/users are (over)loading the filesystem?

• Check applications
  • Why are the processes (over)loading the filesystem?
Assumptions

• Examples assume Lustre filesystem version 1.8
• No apparent errors on
  • Interconnects
  • Clients
  • Servers
  • Storage backends
Checking Lustre servers

• Find out what keeps Lustre servers busy
• Necessary information readily available in stats files
• No problem, case closed
Checking Lustre servers

- Find out what keeps Lustre servers busy
- Necessary information readily available in stats files
- No problem, case closed
- Except for information overload
- Example from a production environment

**MDS**

```
# find /proc/fs/lustre -name "*stats*" | wc -l
2499
```

(9 files/MDS + 4 files/MDT + 3 files/OST + 4 files/client)

**OSS**

```
# find /proc/fs/lustre -name "*stats*" | wc -l
11034
```

(6 files/OSS + 5 files/OST + 3 files/[client*OST])
The Needle in the Haystack

Too many sources to monitor, check, and understand

- Tools like llstat, lstats.sh, ltrack_stats, lustre_req_history, collectl etc. help collecting, but not reducing and interpreting information
- Instead, create artificial load that mimicks typical usage
- Watch statistics to identify relevant files and lines
- Provides small subset of information sources to look at first
- Not comprehensive, but useful for fast initial debugging of common scenarios

- Common troublemakers:
  - `ls -lR`, creating many small files,
  - `rm -rf`, small random i/o,
  - heavy bulk i/o
Metadata loads: `readdir()` + `stat()`

- This and similar types of load are created from recursive filesystem scans like `ls -lR`, `find -newer`, `du -s`, etc.
- Shows up in statistics on MDS

<table>
<thead>
<tr>
<th>Host</th>
<th>File</th>
<th>Counter</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mdt/MDS/mds_readpage/stats</td>
<td>mds_readpage</td>
<td>readdir()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mds_close</td>
<td></td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mdt/MDS/mds/stats</td>
<td>mds_getattr</td>
<td>min/avg/max</td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mds/&lt;fsname&gt;-MDT0000/stats</td>
<td>getattr</td>
<td></td>
</tr>
</tbody>
</table>

- `min/avg/max` make it easier to tell apart unusual load spikes
- Example:

```bash
mds# llstat -i 1 /proc/fs/lustre/mdt/MDS/mds/stats
Name     Cur.Count  Cur.Rate   #Events  Unit    last  min  avg    max    stddev
         mds_getattr 0          0     88727997 [usec]  0     5  22.86  287438 295.06
```
Metadata loads: `readdir()` + `unlink()`

- Typical loads: job cleanup, transfer scripts, `rm -rf`
- Shows up primarily in statistics on MDS

<table>
<thead>
<tr>
<th>Host</th>
<th>File</th>
<th>Counter</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mdt/MDS/mds/stats</td>
<td>mds_readpage</td>
<td>readdir()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mds_close</td>
<td></td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mdt/MDS/mds/stats</td>
<td>mds_reint_unlink</td>
<td>min/avg/max</td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mds/&lt;fsname&gt;-MDT0000/stats</td>
<td>unlink</td>
<td></td>
</tr>
</tbody>
</table>
Metadata loads: file creates

• Typical loads: job output into many small files
• Shows up primarily in statistics on MDS

<table>
<thead>
<tr>
<th>Host</th>
<th>File</th>
<th>Counter</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mds/&lt;fsname&gt;-MDT0000/stats</td>
<td>open</td>
<td>setattr</td>
</tr>
</tbody>
</table>

• There's also a counter called `create`, but creates are (usually) accounted in `open`
## Metadata loads: summary

<table>
<thead>
<tr>
<th>Host</th>
<th>File</th>
<th>Counter</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mdt/MDS/mds_readpage/stats</td>
<td>mds_readpage mds_close</td>
<td><code>ls -R</code></td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mdt/MDS/mds/stats</td>
<td>mds_getattr</td>
<td><code>ls -lR</code></td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mds/&lt;fsname&gt;-MDT0000/stats</td>
<td>getattr</td>
<td><code>ls -lR</code></td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mdt/MDS/mds/stats</td>
<td>mds_reint_unlink</td>
<td><code>rm -rf</code></td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mds/&lt;fsname&gt;-MDT0000/stats</td>
<td>unlink</td>
<td><code>rm -rf</code></td>
</tr>
<tr>
<td>MDS</td>
<td>/proc/fs/lustre/mds/&lt;fsname&gt;-MDT0000/stats</td>
<td>open setattr</td>
<td><code>touch</code></td>
</tr>
</tbody>
</table>
Data I/O: generic statistics

- Read/write I/O statistics accounted by I/O request (per server), and by throughput (per OST)
- Provides general overview
- Cannot distinguish between types of I/O (sequential vs. random, small vs. large I/O request size)
- Example:
  
  ```
  # llobdstat /proc/fs/lustre/obdfilter/aerohpc1-OST0016/stats 1
  
  Timestamp   Read-delta  ReadRate  Write-delta  WriteRate
  1348329721   0.00MB    0.00MB/s   139.00MB    138.85MB/s
  1348329722   0.07MB    0.07MB/s    83.00MB    82.92MB/s
  ```

<table>
<thead>
<tr>
<th>Host</th>
<th>File</th>
<th>Counter</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSS</td>
<td>/proc/fs/lustre/ost/OSS/ost_io/stats</td>
<td>ost_read</td>
<td>min/avg/max</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ost_write</td>
<td></td>
</tr>
<tr>
<td>OSS</td>
<td>/proc/fs/lustre/obdfilter/&lt;fsname&gt;-OSTNNNN/stats</td>
<td>read_bytes</td>
<td>use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>write_bytes</td>
<td>llobdstat</td>
</tr>
</tbody>
</table>
Data I/O: detailed statistics

- Detailed I/O statistics collected per OST allow to identify well-behaved (large, sequential) and ill-behaved (small, random) I/O patterns

<table>
<thead>
<tr>
<th>Host</th>
<th>File</th>
<th>Counter</th>
<th>Ideal</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSS</td>
<td>/proc/fs/lustre/obdfilter/&lt;fsname&gt;-OSTNNNNN/brw_stats</td>
<td>pages per bulk r/w</td>
<td>most RPCs at max</td>
</tr>
<tr>
<td></td>
<td></td>
<td>discontiguous pages/blocks</td>
<td>most RPCs at 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>disk fragmented I/O</td>
<td>most ios at 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>disk I/O size</td>
<td>most ios at 1M</td>
</tr>
</tbody>
</table>

- Read statistics are usually close to ideal (read-ahead), write statistics can reveal ill-behaved I/O
Identifying source of I/O

- Type of problematic I/O should now be known
- Need to find source of I/O
  - Check per-client (per NID) statistics
  - Derive information from RPC request history
- Using RPC request history usually easier
- All per-server stats files accompanied by req_history files providing history of last RPC requests
- RPC history deactivated by default
Using RPC request history

- Activate RPC history by configuring non-zero buffer size, eg.
  
  ```
  # lctl set_param \n  ost.OSS.ost_io.req_buffer_history_max=10240
  ```
  (saves last 10k I/O RPCs on an OSS)

- After a while, read out RPC history, eg.
  
  ```
  # lctl get_param ost.OSS.ost_io.req_history
  ```

- Output format
  
  
  `4134542441:10.1.2.3@o2ib:12345-10.1.2.4@o2ib:x1406392481581555:448:Complete:1348243976:0s(-8s) opc 3`

- Filtering by `opcode` and accounting by `source_nid` reveals client(s) producing the most of the problem RPCs
Matching opcodes to stats

- RPC requests identified by names in stats files, but by numbers in req_history
- Mapping in header `lustre/include/lustre/lustre_idl.h`

<table>
<thead>
<tr>
<th>Opc</th>
<th>Makro</th>
<th>Opc</th>
<th>Makro</th>
<th>Opc</th>
<th>Makro</th>
<th>Opc</th>
<th>Makro</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OST_REPLY</td>
<td>11</td>
<td>OST_OPEN</td>
<td>37</td>
<td>MDS_READPAGE</td>
<td>48</td>
<td>MDS_QUOTACTL</td>
</tr>
<tr>
<td>1</td>
<td>OST_GETATTR</td>
<td>12</td>
<td>OST_CLOSE</td>
<td>38</td>
<td>MDS_CONNECT</td>
<td>49</td>
<td>MDS_GETXATTR</td>
</tr>
<tr>
<td>2</td>
<td>OST_SETATTR</td>
<td>13</td>
<td>OST_STATFS</td>
<td>39</td>
<td>MDS_DISCONNECT</td>
<td>50</td>
<td>MDS_SETXATTR</td>
</tr>
<tr>
<td>3</td>
<td>OST_READ</td>
<td>16</td>
<td>OST_SYNC</td>
<td>40</td>
<td>MDS_GETSTATUS</td>
<td>101</td>
<td>LDLM_ENQUEUE</td>
</tr>
<tr>
<td>4</td>
<td>OST_WRITE</td>
<td>17</td>
<td>OST_SET_INFO</td>
<td>41</td>
<td>MDS_STATFS</td>
<td>102</td>
<td>LDLM_CONVERT</td>
</tr>
<tr>
<td>5</td>
<td>OST_CREATE</td>
<td>18</td>
<td>OST_QUOTACHEK</td>
<td>42</td>
<td>MDS_PIN</td>
<td>103</td>
<td>LDLM_CANCEL</td>
</tr>
<tr>
<td>6</td>
<td>OST_DESTROY</td>
<td>19</td>
<td>OST_QUOTACTL</td>
<td>43</td>
<td>MDS_UNPIN</td>
<td>400</td>
<td>OBD_PING</td>
</tr>
<tr>
<td>7</td>
<td>OST_GET_INFO</td>
<td>33</td>
<td>MDS_GETATTR</td>
<td>44</td>
<td>MDS_SYNC</td>
<td>401</td>
<td>OBD_LOG_CANCEL</td>
</tr>
<tr>
<td>8</td>
<td>OST_CONNECT</td>
<td>34</td>
<td>MDS_GETATTR_NAME</td>
<td>45</td>
<td>MDS_DONE_WRITING</td>
<td>402</td>
<td>OBD_QC_CALLBACK</td>
</tr>
<tr>
<td>9</td>
<td>OST_DISCONNECT</td>
<td>35</td>
<td>MDS_CLOSE</td>
<td>46</td>
<td>MDS_SET_INFO</td>
<td>(....)</td>
<td>(....)</td>
</tr>
<tr>
<td>10</td>
<td>OST_PUNCH</td>
<td>36</td>
<td>MDS_REINT</td>
<td>47</td>
<td>MDS_QUOTACHEK</td>
<td>(....)</td>
<td>(....)</td>
</tr>
</tbody>
</table>
Client-side statistics

- With client NID known, need to find process that is I/O source
- Usually easy task on cluster nodes running single/few jobs
- Much harder on large multi-user systems
- Determine PIDs with active Lustre I/O, (ab)using extents stats
- Limit statistics to subset of processes

<table>
<thead>
<tr>
<th>Host</th>
<th>File</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client</td>
<td>/proc/fs/lustre/llite/&lt;fsname&gt;-&lt;uuid&gt;/extents_stats_per_process</td>
<td>PIDs with active I/O</td>
</tr>
<tr>
<td>Client</td>
<td>/proc/fs/lustre/llite/&lt;fsname&gt;-&lt;uuid&gt;/stats</td>
<td>Generic client-side stats</td>
</tr>
<tr>
<td>Client</td>
<td>/proc/fs/lustre/llite/&lt;fsname&gt;-&lt;uuid&gt;/stats_track_pid</td>
<td>Limit stats to PID</td>
</tr>
<tr>
<td>Client</td>
<td>/proc/fs/lustre/llite/&lt;fsname&gt;-&lt;uuid&gt;/stats_track_ppid</td>
<td>Limit stats to PPID</td>
</tr>
<tr>
<td>Client</td>
<td>/proc/fs/lustre/llite/&lt;fsname&gt;-&lt;uuid&gt;/stats_track_gid</td>
<td>Limit stats to GID</td>
</tr>
</tbody>
</table>
Simple I/O profiles

Never attribute to the filesystem that which is adequately explained by application stupidity.

- Trace slow or misbehaving applications
- Try to determine where most of I/O time is spent
- `strace -T` is a great tool to easily obtain simple I/O profiles
- Practical example: Slow checkout of large SVN repository
I/O profile

aggregate runtime per syscall

Solaris NFS4
Solaris NFS4+Fix
Filesystem profiles

![Graph showing filesystem profiles for various file operations and different file systems.]

- ext3
- NFSv3 Linux
- NFSv3 Solaris
- NFSv4 Solaris

Operations include: access, chmod, close, fcntl, fstat, getcwd, getdents, lseek, lstat, mkdir, open, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open_RDONLY, open ורקд

page 21
Conclusion

• Lustre provides huge amount of profiling information
• Pinpointing the right information can be challenging to the uninitiated
• Pre-defined subset of information useful for monitoring, or initial checks to identify most common usage scenarios
• Simple application profiling can reveal sub-optimal I/O patterns
• Optimising applications can be more effective than filesystem tuning
Thank you!

Daniel Kobras
science + computing ag
www.science-computing.de
www.hpc-wissen.de
Telefon 07071 9457-0
info@science-computing.de