I/O Prefetch Cache as QEMU Block Filter Driver

Pavel Butsykin
Virtuozzo, inc.
KVM Forum 2016
Prefetch cache

• Idea – Fetch the data before it is needed
• There are different types of prefetch cache that focus on different read patterns
• Prefetching sequential data read (read-ahead) is one of the types of such cache
Read-ahead

- Random access
  - can displace a lot of cache
  - can lead to performance degradation

- Sequential read detection?
QEMU Block Filter Driver

Guest

virtio
sata
ide ...

pcache

qcow2
parallels
vmdk ...

raw

Host fs
Parallel and sequential read

Sequential stream 1
1 2 3 4 5 6

Sequential stream 2
A B C D E F

Sequential stream 3
g h i j k l

One of the possible sequences
1 A B g 2 C

pcache
Sequential read detection

Node 124K - 1M

Pool requests

RBTree key

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32K</td>
</tr>
<tr>
<td>112K</td>
<td>8K</td>
</tr>
<tr>
<td>4M</td>
<td>8K</td>
</tr>
</tbody>
</table>

128K

512K

630K

120K 4K

64K 10K

124K 1M

FIFO

Request
PCache AIO read overview

• Skip large requests (by default larger than 64Kb)
• Update request statistics
• Cache lookup
  • hit
  • partial hit
  • miss
• Read-ahead
  • check request sequence
  • read into cache a chunk of data from the end of the current request
Cache memory

- The cache memory has limited size (4Mb by default)
- The cache is managed by LRU algorithm
Why you need LRU for the prefetch cache?

If you have read one part of the node, then there is a high probability that you will soon read the remaining parts of the node.
Partial cache hit

Read request 1:
- Offset: 0; Size: 420K

Read request 2:
- Offset: 600K; Size: 130K

Part of cache memory:
- Offset: 412K; Size: 8K
- Offset: 600K; Size: 6K
- Offset: 632K; Size: 21K
- Offset: 673K; Size: 29K

HDD
PCache AIO write overview

• Drop all nodes intersecting with request
• Write-through
Rescheduling AIO requests

What to do if the requested node is in-flight?

Offset; size; qiov; node_list; refcount; ...

key; request_list; refcount; status; readcount; ...

request

node

many-to-many

new

wait

node3

node2

node1
PCache AIO request complete

How to complete an request which expects other requests?

- request complete
- 1 node complete
- 2 request unreference
- N request unreference

- node complete
- refcount=0

- death
- refcount=0

- complete!
Read-ahead policy

• Original requests are not written to the cache and only serve to update statistics
• Filtering of large requests helps to detect sequential read
• If part of the readahead is already in the cache then only the missing pieces will be fetched from disk
<table>
<thead>
<tr>
<th>#</th>
<th>I/O Test</th>
<th>SSDSC2BW120A4 EXT4</th>
<th>QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM</th>
<th>QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM + dataplane</th>
<th>QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM + pcache</th>
<th>QEMU 2.6.50 VirtIO qcow2 Linux 4.4.0 Fedora-22 2SMP 2GB VM + dataplane + pcache</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2G-read-seq-4K(01)</td>
<td>100%</td>
<td>+10.7%</td>
<td>+341%</td>
<td>+298%</td>
<td>+490%</td>
</tr>
<tr>
<td>2</td>
<td>2G-read-seq-4K(04)</td>
<td>100%</td>
<td>+5.0%</td>
<td>+231%</td>
<td>+215%</td>
<td>+225%</td>
</tr>
<tr>
<td>3</td>
<td>2G-read-seq-4K(16)</td>
<td>100%</td>
<td>+0.1%</td>
<td>+80.3%</td>
<td>+80.1%</td>
<td>+72.1%</td>
</tr>
<tr>
<td>4</td>
<td>2G-read-seq-4K-AIO4(01)</td>
<td>100%</td>
<td>+43.3%</td>
<td>+190%</td>
<td>+102%</td>
<td>+191%</td>
</tr>
<tr>
<td>5</td>
<td>2G-read-seq-4K-AIO4(04)</td>
<td>100%</td>
<td>+1.4%</td>
<td>+79.0%</td>
<td>+76.6%</td>
<td>+78.2%</td>
</tr>
<tr>
<td>6</td>
<td>2G-read-seq-4K-AIO4(16)</td>
<td>100%</td>
<td>+0.6%</td>
<td>+77.6%</td>
<td>+76.5%</td>
<td>+70.3%</td>
</tr>
<tr>
<td>7</td>
<td>2G-read-seq-4K-AIO32(01)</td>
<td>100%</td>
<td>-7.8%</td>
<td>-0.8%</td>
<td>+7.6%</td>
<td>+1.2%</td>
</tr>
<tr>
<td>8</td>
<td>2G-read-seq-4K-AIO32(04)</td>
<td>100%</td>
<td>-4.4%</td>
<td>+21.8%</td>
<td>+27.4%</td>
<td>+24.9%</td>
</tr>
<tr>
<td>9</td>
<td>2G-read-seq-4K-AIO32(16)</td>
<td>100%</td>
<td>-3.0%</td>
<td>+46.4%</td>
<td>+50.8%</td>
<td>+46.7%</td>
</tr>
<tr>
<td>10</td>
<td>2G-read-rnd-4K(01)</td>
<td>100%</td>
<td>+1.1%</td>
<td>+3.0%</td>
<td>+1.9%</td>
<td>+1.9%</td>
</tr>
<tr>
<td>11</td>
<td>2G-read-rnd-4K(04)</td>
<td>100%</td>
<td>+2.5%</td>
<td>+0.4%</td>
<td>-2.0%</td>
<td>+1.7%</td>
</tr>
<tr>
<td>12</td>
<td>2G-read-rnd-4K(16)</td>
<td>100%</td>
<td>+7.1%</td>
<td>+5.4%</td>
<td>-1.6%</td>
<td>+5.7%</td>
</tr>
<tr>
<td>13</td>
<td>2G-read-rnd-4K-AIO4(01)</td>
<td>100%</td>
<td>+1.2%</td>
<td>-0.8%</td>
<td>-1.9%</td>
<td>+1.5%</td>
</tr>
<tr>
<td>14</td>
<td>2G-read-rnd-4K-AIO4(04)</td>
<td>100%</td>
<td>+2.4%</td>
<td>+6.9%</td>
<td>+4.3%</td>
<td>+7.2%</td>
</tr>
<tr>
<td>15</td>
<td>2G-read-rnd-4K-AIO4(16)</td>
<td>100%</td>
<td>-0.5%</td>
<td>+4.9%</td>
<td>+5.4%</td>
<td>+4.1%</td>
</tr>
<tr>
<td>16</td>
<td>2G-read-rnd-4K-AIO32(01)</td>
<td>100%</td>
<td>+1.4%</td>
<td>+0.1%</td>
<td>-1.3%</td>
<td>+1.5%</td>
</tr>
<tr>
<td>17</td>
<td>2G-read-rnd-4K-AIO32(04)</td>
<td>100%</td>
<td>+2.9%</td>
<td>+0.4%</td>
<td>-2.5%</td>
<td>+3.1%</td>
</tr>
<tr>
<td>18</td>
<td>2G-read-rnd-4K-AIO32(16)</td>
<td>100%</td>
<td>-4.6%</td>
<td>-0.5%</td>
<td>+4.3%</td>
<td>+0.6%</td>
</tr>
</tbody>
</table>
# Test Scores vs #1

<table>
<thead>
<tr>
<th>#</th>
<th>Test</th>
<th>Scores</th>
<th>Scores</th>
<th>vs #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dir_readdir</td>
<td>134</td>
<td>137</td>
<td>+1.9%</td>
</tr>
</tbody>
</table>

Testcase: create a directory and populate it with 10 subdirs and 10 files with max depth 3 once before test. Total: there are 10^3 dirs and 10^3 files. All files are empty:

1. open() root dir, then readdir() recursively, close():

```
Qemu bench

```bash
$ ./qemu-img bench -d 1 -c 262144 -f qcow2 -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 1 in parallel (starting at offset 0, step size 4096)
Run completed in 19.594 seconds.

$ ./qemu-img bench -d 1 -c 262144 -f pcache -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 1 in parallel (starting at offset 0, step size 4096)
Run completed in 4.378 seconds.

$ ./qemu-img bench -d 8 -c 262144 -f qcow2 -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 8 in parallel (starting at offset 0, step size 4096)
Run completed in 5.933 seconds.

$ ./qemu-img bench -d 8 -c 262144 -f pcache -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 8 in parallel (starting at offset 0, step size 4096)
Run completed in 4.356 seconds.

$ ./qemu-img bench -d 64 -c 262144 -f qcow2 -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 64 in parallel (starting at offset 0, step size 4096)
Run completed in 4.659 seconds.

$ ./qemu-img bench -d 64 -c 262144 -f pcache -s 4096 -S 4096 -t none ./image.qcow2
Sending 262144 read requests, 4096 bytes each, 64 in parallel (starting at offset 0, step size 4096)
Run completed in 4.204 seconds.
```
4K AIO read requests (pcache)

For readahead_size = 64Kb, it was the expected result
4K AIO read requests (pcache + iothread)

iothread helped to merge a lot of requests
Conclusions

• PCache can optimize certain I/O patterns without pessimizing others
• PCache implementation in the form of the driver filter is unintrusive
• PCache is not universally useful, benchmark your patterns before enabling
Questions?