POSTCOPY PREEMPTION
OUTLINES

- A quick re-cap on live migration
- Postcopy limitations, challenges
- Three optimizations
  - Channel separation, huge page, thread model
- Performance Results
- Future works
WHAT IS POSTCOPY?

- Allows the VM to start running with partial RAM (compared to precopy)
- Trap page faults when page missing (userfaultfd)
- Always converges
WHAT IS POSTCOPY PREEMPTION?

- A new capability ("postcopy-preempt") introduced for postcopy-only
  - Need to be enabled on both src/dst QEMU
  - Not compatible with vanilla postcopy
  - No extra configuration needed
- Direct performance improvement on the speed of handling page faults
  - More test results at the end
LIVE MIGRATION (PRECOPY)

Page Stream

Dirty Page
Clean Page
Missing Page
Stall Page
LIVE MIGRATION (PRECOPY COMPLETED)
LIVE MIGRATION (PRECOPY COMPLETED)

SRC VM

DST VM

STOPPED

RUNNING
LIVE MIGRATION (PRECOPY, BUT RUN ON DST?)

SRC VM

DST VM

Page Stream

Dirty Page

Clean Page

Missing Page

Stall Page

RUNNING

STOPPED
LIVE MIGRATION (POSTCOPY)

SRC VM

DST VM

Page Stream

Messages

Clean Page

Missing Page

Page Requests

RUNNING

STOPPED
LIVE MIGRATION (POSTCOPY)

SRC VM

DST VM

Page Stream

Messages

Urgent Page

Background Page

Missing Page

Page Requests

STOPPED

RUNNING
POSTCOPY LIMITATIONS

- Split brain, e.g. network failures during postcopy
  - Postcopy recovery (since QEMU v3.0.0)
- High page request latency
- For huge pages...
  - Hugetlb double map allows page to be mapped in PAGE_SIZE
    https://lore.kernel.org/all/20220624173656.2033256-1-jthoughton@google.com/
- Page transfers are slow even for small pages for QEMU
  - An average of 12ms on directly attached 10Gbps network for random access
ISSUE 1 - BACKGROUND PAGE FLUSH

MIGRATION THREAD

PAGE QUEUE

RETURN THREAD

BLOCKED BY BACKGROUND PAGES

Page Stream

PAGE RESOLVE

PAGE FAULT HANDLER

Messages

Urgent Page
Background Page
Page Requests

Page Requests
SOLUTION 1 - CHANNEL SEPARATION

- MIGRATION THREAD
- PAGE QUEUE
- RETURN THREAD
- Background Page Stream
- Urgent Page Stream
- Messages
- Page Requests
- Page Resolve
- Fast Page Resolve
- Page Fault Handler

- Urgent Page
- Background Page
ISSUE 2 - HUGE PAGE GRANULE

- QEMU sends pages always in huge page granule
  - Before finish sending one huge page, we cannot send another page
  - An urgent page cannot preempt sending of a background huge page
- Why?
  - QEMU receiving page using temp huge page buffers, which are limited
ISSUE 2 - HUGE PAGE GRANULE

- **MIGRATION THREAD**
  - Background Page Stream
    - Pages: 10, 11, 12
  - Urgent Page Stream
    - Pages: 5, 6, 7, 8

- **PAGE RESOLVE**
- **FAST PAGE RESOLVE**

**BLOCKED BEFORE SENDING WHOLE HOST BACKGROUND HOST PAGE**

- Urgent Page
- Background Page
SOLUTION 2 - HUGE PAGE PREEMPTIONS

MIGRATION THREAD

Background Page Stream

Urgent Page Stream

PAGE RESOLVE

FAST PAGE RESOLVE

SEND URGENT PAGE RIGHT AWAY

Urgent Page

Background Page
ISSUE 3 – MIGRATION THREAD ITSELF!

- "migration_thread" is the thread to save VM on src QEMU
  - Background sendmsg() blocks not only itself but all the rest (e.g. sending urgent page)
- The only thread to migrate a guest page, due to
  - Legacy state maintenances (RAMState, PageSearchState, bitmaps, etc.)
  - Required by all kinds of features (compression, XBZRLE, multifd, etc.)
    - Compression: distribute raw pages to compressor threads
    - XBZRLE: global xbzrle state maintenance
    - Multifd: entrance of page distributions
SOLUTION 3 - ???

- Refactor global states into per-channel ones
  - Turning PageSearchStatus into a per-channel structure, one for each channel
- Manage page ownerships, aka:
  When there are >1 threads sending, who should send which page?
  - Who took the bitmap bit (protected by bitmap_mutex)
  - Make sure to release any global lock during sending (e.g. sendmsg() could block)
- Send pages outside migration_thread
  - How about... the return thread????
- Drop page request queue, because we don't need it anymore!
(A RECAP ON PREVIOUS . . .)

- Migration Thread
- Page Queue
- Return Thread
- Background Page Stream
- Urgent Page Stream
- Page Resolve
- Fast Page Resolve
- Page Fault Handler
- Messages
- Urgent Page
- Background Page
- Page Requests
SOLUTION 3 - REUSE RETURN THREAD TO SEND PAGES

Background Page Stream

Urgent Page Stream

Messages

Urgent Page

Background Page

Page Requests
PERFORMANCE NUMBERS

- VM: 20 cpus, 20GB mem, 1 busy random write workload over 18GB

- Test program: mig_mon mm_dirty -m 18000 -p random
  https://github.com/xzpeter/mig_mon

- Measure average page fault latencies
  https://github.com/xzpeter/small-stuffs/blob/master/tools/huge_vm/uffd-latency.bpf

- Results (~50x speedup in 4K average page request latency)
  - Vanilla: 12093 (us)
  - Preempt Full (solution 1+2+3): 229 (us)
## DISTRIBUTIONS OF LATENCIES

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Vanilla Average: 12093 (us) @delay_us:</th>
<th>Preempt Full Average: 229 (us) @delay_us:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[1]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>[2, 4]</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>[4, 8]</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>[8, 16]</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>[16, 32]</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>[32, 64]</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>[64, 128]</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>[128, 256]</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>[256, 512]</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>[512, 1K]</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>[1K, 2K]</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>[2K, 4K]</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>[4K, 8K]</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>[8K, 16K]</td>
<td>2397</td>
</tr>
<tr>
<td></td>
<td>[16K, 32K]</td>
<td>7</td>
</tr>
</tbody>
</table>
FUTURE WORK

- Postcopy preempt part 1 merged in v7.1.0 (including solution 1+2)
  https://lore.kernel.org/qemu-devel/20220707185342.26794-1-peterx@redhat.com/

- Postcopy preempt part 2 RFC posted (including solution 3), during review
  https://lore.kernel.org/qemu-devel/20220829165659.96046-1-peterx@redhat.com/

- Comments welcomed