Lazy Process Migration

Adrian Reber <areber@redhat.com>
Mike Rapoport <rppt@linux.vnet.ibm.com>

Linux Plumbers Conference 2016
November 1-4, Santa Fe, New Mexico, USA

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 688386
Background

Implementation

Future Plans
Process Downtime During Migration

![Diagram showing migration times for SSD and RAM drive backed migrations. The graph plots migration time in seconds against size in GB of migrated process. SSD backed migration (IB) has the steepest line, followed by RAM drive backed migration (IB), and then RAM drive backed migration. SSD backed migration (IB) has the longest migration time at 800 seconds for 50 GB of data.](image-url)
Process Downtime During Migration

Initiate Migration → Quiesce Process → Finish Migration → Resume Process

Source System, Destination System

Figure: Process Migration
Optimizations - Pre-Copy

Figure: Pre-Copy Migration
Possible Drawbacks Using Pre-Copy

Lazy Process Migration

Migration time in seconds

<table>
<thead>
<tr>
<th>State of application</th>
<th>Migration time without pre-copy</th>
<th>Migration time after pre-copy</th>
<th>Pre-copy duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>3s</td>
<td>4s</td>
<td>2s</td>
</tr>
<tr>
<td>Stabilization</td>
<td>3s</td>
<td>3s</td>
<td>22s</td>
</tr>
<tr>
<td>Calculation</td>
<td>3s</td>
<td>8s</td>
<td>42s</td>
</tr>
</tbody>
</table>

Transferred memory in GB

<table>
<thead>
<tr>
<th>State of application</th>
<th>Transfer size without pre-copy</th>
<th>Second transfer</th>
<th>Transfer size pre-copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>5.2GB</td>
<td>7.5GB</td>
<td>5.2GB</td>
</tr>
<tr>
<td>Stabilization</td>
<td>7.5GB</td>
<td>7.5GB</td>
<td>7.5GB</td>
</tr>
<tr>
<td>Calculation</td>
<td>9.7GB</td>
<td>9.7GB</td>
<td>2.8GB</td>
</tr>
</tbody>
</table>
Optimizations - Post-Copy

Figure: Post-Copy Migration
CRIU And Userfaultfd

- Userfaultfd (UFFD) integration into CRIU
- Most pages can be handled by UFFD
  - Anonymous private mappings are already supported
  - Shared memory is planned
- Process downtime can be decreased
- To restore a 200MB process
  - transfer 200MB without Post-Copy
  - transfer 116KB with Post-Copy
Lazy Migration Details 1

Source (dump)

- Memory pages are marked as lazy during dump
- Lazy memory pages are not written to disk
- Source system waits for requests to transfer lazy memory pages via TCP

Destination (restore)

- CRIU registers memory areas with userfaultfd and connects to the source
- The process is restored with no memory
- Process accesses to memory generate page faults which are handled by the UFFD daemon
Lazy Migration Details 2

criu lazy-pages (uffd daemon)

criu restore

/path/to/lazy-pages.socket

PID

UFFD

destination system
Lazy Migration Details 3

- Access memory
- Mark pages as lazy
- Jump into restored process
- Restore process
- CRIU restore

Diagram:
- Kernel
- Restored process
- CRIU restore
- Access memory
- Mark pages as lazy
- Jump into restored process
- Restore process
criu page-server
criu lazy-pages
uffd daemon
kernel
restored process

access memory

request page
via uffd

request page
via tcp

transfer page
via tcp

transfer page
via uffd

resume process
Current status

- In criu-dev branch:
  - local lazy restore works
  - remote lazy restore works
  - combination of pre-copy and post-copy works
- Kernel patches for userfaultfd\(^1\) are under review on linux-mm\(^2\)
  - non-cooperative mode (support for `fork()` and other events)
  - support for shared memory

\(^1\)https://git.kernel.org/cgit/linux/kernel/git/andrea/aa.git/aa.git
\(^2\)http://www.spinics.net/lists/linux-mm/msg115992.html
Limitations

- A process that executes `fork()`, `madvise(MADV_DONTNEED)` or `mremap` will fail
- Shared (tmpfs) and hugeltbfs mappings cannot be handled by userfaultfd
- Post-copy performance is far from optimal
Future plans

- Add post-copy support to `p.haul`, `runc`, `lxc`
- Non-cooperative `userfaultfd` (`fork()` and other events) in CRIU and in the kernel
- Shared memory post-copy
- Nested `userfaultfd`
- Optimizations
The end.

Thanks for listening.