Better Live Migration

Xiao Guangrong
<xiaoguangrong@tencent.com>

Agenda

- Background
- Shortcomings and solutions
- Status

Background

- Live migration plays a very important role in industry
  - The infrastructure of load balancer
  - Error recovery
  - Software & hardware upgrade
- Live migration is challengeable in the production
  - Memory intensive workload in VM
  - The rate that vCPU dirties memory is far more faster than networking
  - IO sensitive VM requires extreme low downtime and low latency to handle IO requests

Background (Cont.)

- QEMU/KVM gains some features to improve live migration
  - Compression, XBZRLE, auto-converge, etc.
- However, none of them works perfectly
- We, Tencent cloud, are continually improving live migration on our productions
  - We introduced “fast write protection” on KVM Forum 2017
  - In this presentation, we focus on the improvements of existing features in QEMU/KVM

Shortcomings and solutions

- Compression
- XBZRLE
- Auto-converge

Compression

- Use multithreads to compress the data before put it on the network to reduce transferred size
  - Data should be compressible
  - CPU intensive, the system should have enough resources to do compression
- Shortcomings
  - User has no way to check if compression works well
  - Inefficient multithread model
    - Multiple locks
    - Too many waits & wakeups

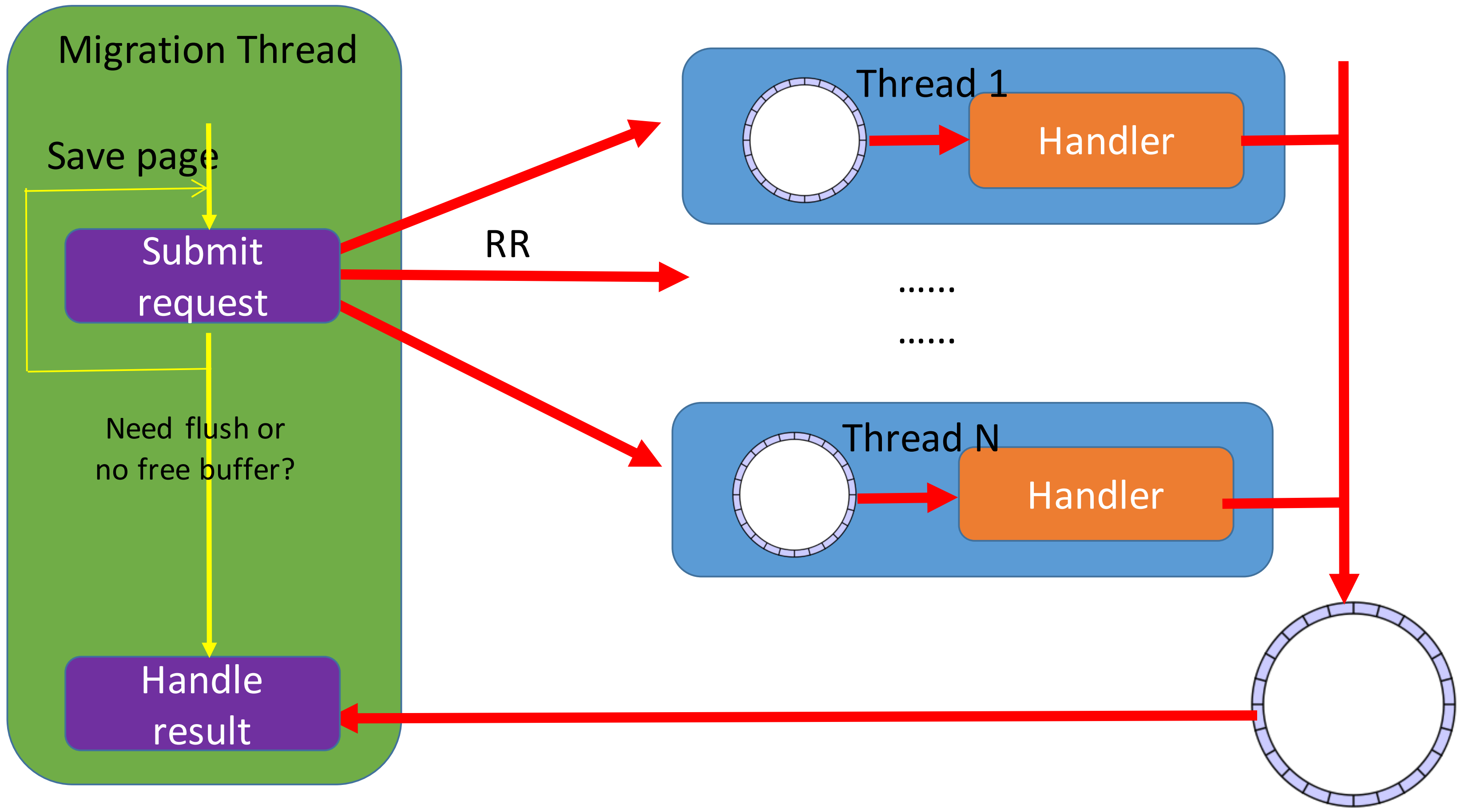
Compression: solutions

- We collect the statistics and show them to user
  - Compress-rate, Busy-rate, etc.
- User can adjust the parameters based on these statistics
  - Compress level
  - Threads number
  - Etc.

```
info migrate
globals:
store-global-state: on
only-migratable: off
send-configuration: on
send-section-footer: on
….........
Migration status: active
total time: 1019540 milliseconds
expected downtime: 2263 milliseconds
setup: 218 milliseconds
transferred ram: 252419995 kbytes
throughput: 2469.45 mbps
remaining ram: 15611332 kbytes
total ram: 62931784 kbytes
duplicate: 915323 pages
skipped: 0 pages
normal: 59673047 pages
normal bytes: 238692188 kbytes
dirty sync count: 28
page size: 4 kbytes
dirty pages rate: 170551 pages
compression pages: 121309323 pages
compression busy: 60588337
compression busy rate: 0.36
compression reduced size: 484281967178
compression rate: 0.97
```

Compression: solutions (Cont.)

- We introduced a lockless multithread model



Compression: solutions (Cont.)

- Performance result
  - Host: Xeon(R) Gold 6142 CPU @ 2.60GHz * 64 + 256G; VM: 16 vCPUs and 60G, repeatedly write memory in it
  - Use 16 threads to compress and decompress
  - CPU usage

Before

	Main Thread	[De]Compress Threads
Source	96%	some use 45%, others are very low ~6%
Dest.	96%	some use 58%, other are very low ~10%

After

	Main Thread	[De]Compress Threads
Source	60%	60%
Dest.	100%	75%

Compression: solutions (Cont.)

- Migration result

  Before: Cannot complete

  After

```
Migration status: completed
total time: 64118 milliseconds
downtime: 29 milliseconds
setup: 223 milliseconds
transferred ram: 13345135 kbytes
throughput: 1705.10 mbps
remaining ram: 0 kbytes
total ram: 62931784 kbytes
duplicate: 574921 pages
skipped: 0 pages
normal: 2570281 pages
normal bytes: 10281124 kbytes
dirty sync count: 9
page size: 4 kbytes
compression pages: 28007024 pages
compression busy: 3145182
compression busy rate: 0.08
compression reduced size: 111829024985
compression rate: 0.97
```

XBZRLE (Xor Binary Zero Run-Length-Encoding)

- "Instead of sending the changed guest memory page this solution will send a compressed version of the updates"
  - Works only if the data is friendly to XBZRLE
- Need extra memory to save the previous memory pages
- Shortcomings
  - User has no way to check if XBZRLE works well
  - XBZRLE is a CPU sensitive workload and it slows down the whole process

XBZRLE: solutions

- Collect the statistics and show them to user
  - Data reduced rate
- Use multithreads to do XBZRLE
  - Based on lockless multithreads model

```
info migrate
globals: store-global-state=1, only_migratable=0, send-
……
……
Migration status: active
total time: 11825292 milliseconds
expected downtime: 5492062 milliseconds
setup: 244 milliseconds
transferred ram: 543826215 kbytes
throughput: 4.98 mbps
remaining ram: 24306832 kbytes
total ram: 62931784 kbytes
duplicate: 7980101 pages
skipped: 0 pages
normal: 135634947 pages
normal bytes: 542539788 kbytes
dirty sync count: 3381
page size: 4 kbytes
dirty pages rate: 634526 pages
cache size: 34359738368 bytes
xbzrle transferred: 147029 kbytes
xbzrle pages: 1084010 pages
xbzrle cache miss: 127544642
xbzrle cache miss rate: 0.00
xbzrle overflow : 651
xbzrle reduce size : 4301477037
xbzrle reduce rate: 1.00
```

XBZRLE: solutions (Cont.)

- Performance result

Before (Migration can not complete)

```
globals: store-global-state=1, only_migratable=0, send-
...
Migration status: active
total time: 11825292 milliseconds
expected downtime: 5492062 milliseconds
setup: 244 milliseconds
transferred ram: 543826215 kbytes
throughput: 4.98 mbps
remaining ram: 24306832 kbytes
total ram: 62931784 kbytes
duplicate: 7980101 pages
skipped: 0 pages
normal: 135634947 pages
normal bytes: 542539788 kbytes
dirty sync count: 3381
page size: 4 kbytes
dirty pages rate: 634526 pages
cache size: 34359738368 bytes
xbzrle transferred: 147029 kbytes
xbzrle pages: 1084010 pages
xbzrle cache miss: 127544642
xbzrle cache miss rate: 0.00
xbzrle overflow : 651
xbzrle reduce size : 4301477037
xbzrle reduce rate: 1.00
```

After (complete even if use half of memory than before)

```
globals: store-global-state=1,  only_migratable=0, send-
...
Migration status: completed
total time: 400307 milliseconds
downtime: 79 milliseconds
setup: 214 milliseconds
transferred ram: 128504027 kbytes
throughput: 2629.76 mbps
remaining ram: 0 kbytes
total ram: 62931784 kbytes
duplicate: 7665569 pages
skipped: 0 pages
normal: 32045609 pages
normal bytes: 128182436 kbytes
dirty sync count: 30
page size: 4 kbytes
cache size: 34359738368 bytes
xbzrle transferred: 3802 kbytes
xbzrle pages: 70072728 pages
xbzrle cache miss: 11757676
xbzrle cache miss rate: 0.00
xbzrle overflow : 0
xbzrle reduce size : 287014183873
xbzrle reduce rate: 1.00
```

Auto-converge

- It dynamically throttles vCPUs to force the VM to dirty less memory
- Continually increase the amount of guest cpu throttling until guest memory write speed slows enough
- Shortcomings
    - It make VM completely unusable if live migration is still unsuccessful
    - Big latency to handle IO request, e.g, packet loss, ping test failure, etc.
    - So, it can not work for CPU and IO sensitive VMs

Auto-converge: solutions

- Introduce the x-cpu-throttle-max, it specifies min. capability the vCPU can use
- Throttle vCPUs based on the IO statistics…

Status

- Some optimizations of compression have been merged to QEMU upstream
- Lockless multithreads model has been reviewing in the community
- Others are ready to be pushed out

Q/A?

Reference

- QEMU source code
  - https://git.qemu.org/?p=qemu.git;a=summary
- Compression
  - https://git.qemu.org/?p=qemu.git;a=blob_plain;f=docs/multi-thread-compression.txt
- XBZRLE
  - https://git.qemu.org/?p=qemu.git;a=blob;f=docs/xbzrle.txt
- Auto-coverge
  - https://wiki.qemu.org/Features/AutoconvergeLiveMigration