KVM Live Migration Optimization

Li, Liang Zhang, Yang

Aug 2015

Agenda

- Background
- Problems
- Solutions
- Performance
- Work in progress

Background

- Live migration usage in cloud computing
  - facilitate maintenance
  - load balancing
  - energy saving
- Goals
  - Reduce total live migration time
  - Reduce VM down time
  - Improve migration successful ratio
- Existing optimizations
  - RDMA
  - XBZRLE
  - Auto convergence

Problems

- Network bandwidth could be the bottle neck
  - Network is usually shared.
  - 1Gbps Network is still widely used.
  - Geographic migration
- Low efficient data processing in ram bulk stage
  - Unused pages can be skipped
  - Free pages can be skipped
  - The transmission of zero pages can be skipped
- Time consuming operation in pause and copy stage
  - migration_end
  - blk_mig_cleanup

Solutions

- Multiple thread (de)compression
- Skip the unused pages in ram bulk stage
- Delay the non-emergency operations

Multiple thread (de)compression

- Time spend on different stages
  - Most of the time is spent on sending data if the network bandwidth is low

- Time spend on different stages when using compression
  - Compression can help to reduce the data traffic, and decrease time spend on sending data
  - Compression takes extra time

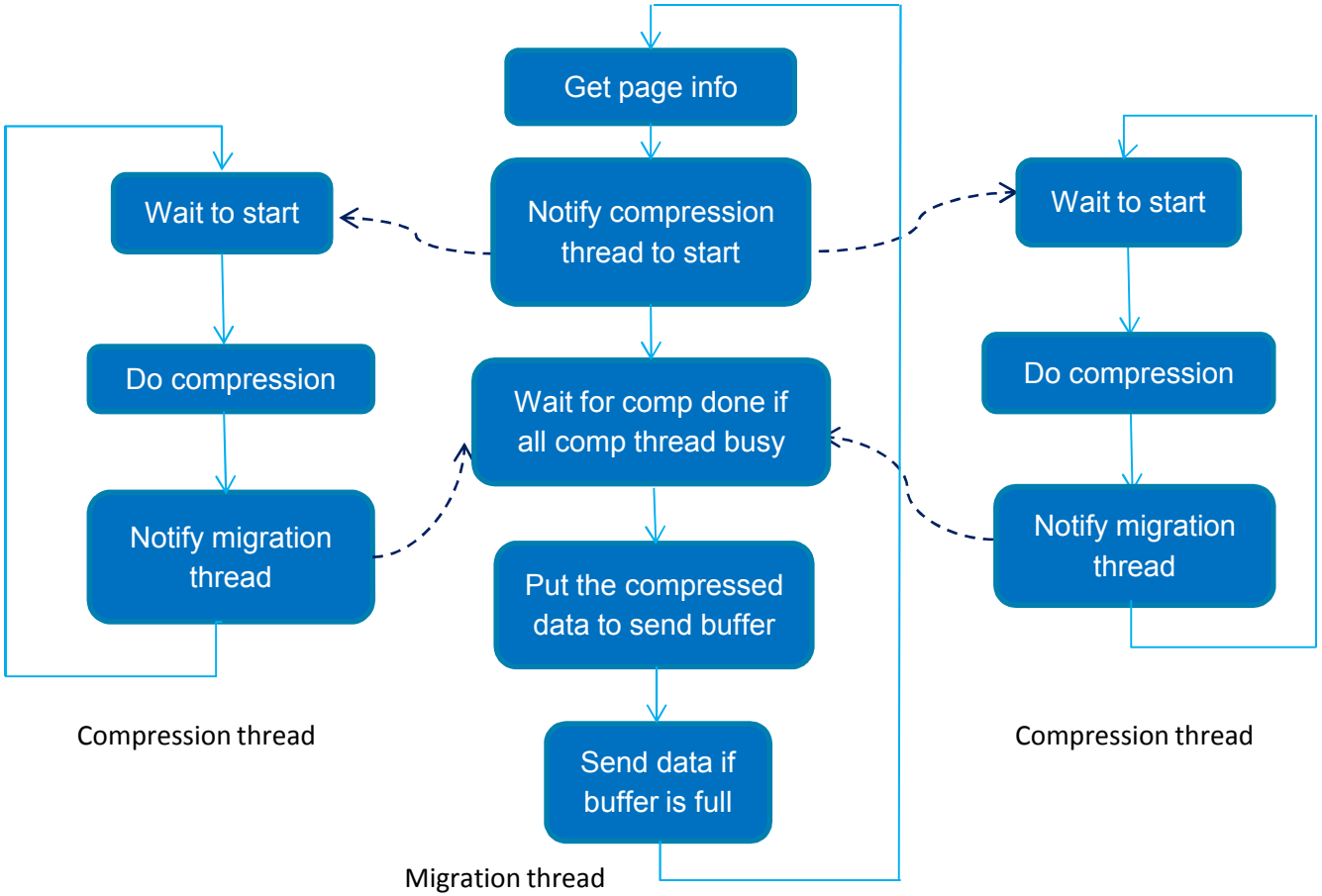
  - Multiple thread is used to accelerate the (de)compression process

Multiple thread (de)compression

- Multiple thread (de)compression is a new live migration feature
  - Instead of sending the guest memory directly, this solution compresses the RAM page before sending.
  - Have been merged into QEMU 2.4.0.

- Relation ship between multiple thread (de)compression and XBZRLE
  - Both aim for reduce the data traffic in network
  - XBZRLE compresses the page updates.
  - Multiple thread (de)compression compresses the original page.
  - Multiple thread (de)compression transfers compressed data in the ram bulk stage, XBZRLE can't do that.
  - Multiple thread co-work with XBZRLE can minimize the data traffic in theory.
  - Multiple thread only takes effect in the ram bulk stage if co-work with XBZRLE.

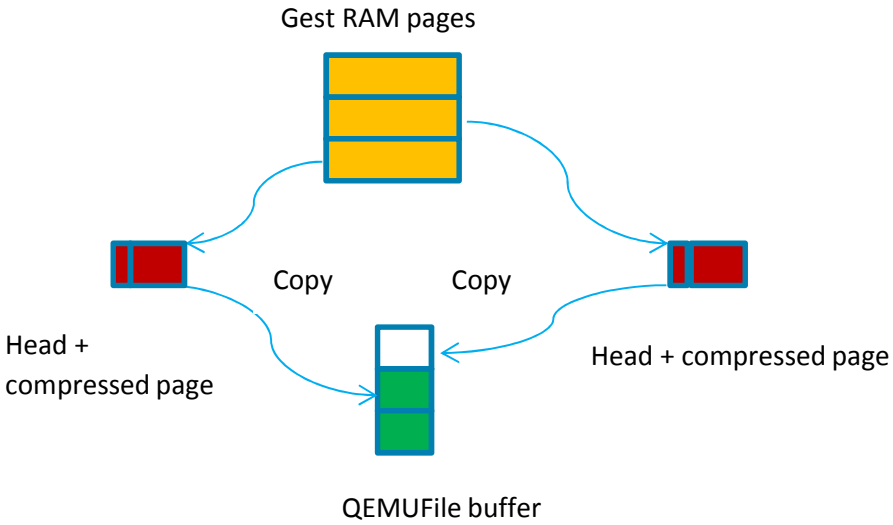
Multiple thread (de)compression details



The relation ship between migration thread and compression threads

Multiple thread (de)compression details

- About data copy
  - Data copy happened when putting the compressed page to QEMUFile



- About page sequence
  - In the block range, the sequence of the page is no matter
  - If a new block begins, all the pages belong to the previous block should be send out first.

Offload the overhead from CPU

- About the CPU usage?
  - 760% on source side
  - 50% on the source side when use the original implementation.
- Solutions
  - Use some faster compression algorithm, like Quicklz, LZ4.

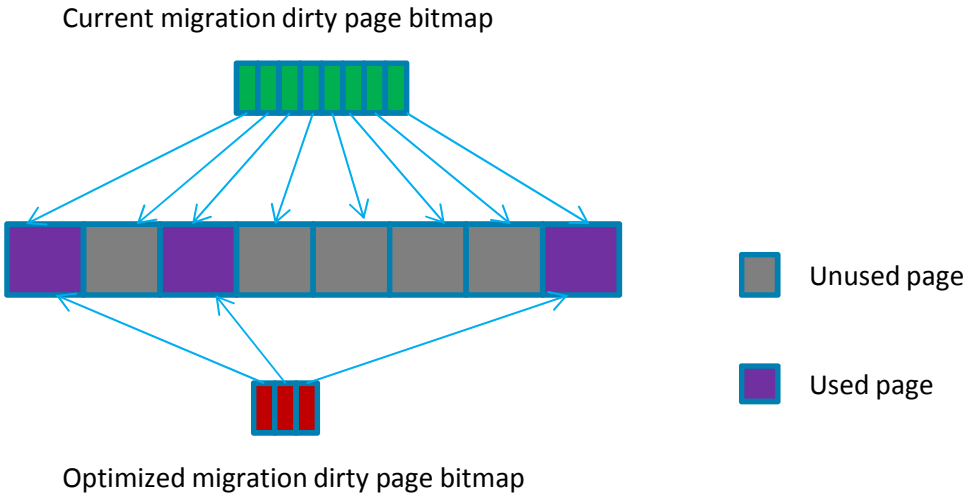
	Zlib 8 threads	LZ4 8 threads	No compression
CPU usage	760%	108%	51%
Total migration time (Sec)	20	20	34

  - Use hardware (de)compression accelerator to offload the over head from CPU. CPU usage can be reduced more if using the asynchronous mode of the hardware (de)compression accelerator.

	Zlib	Zlib with hardware accelerator
CPU usage	760%	150%

Skip unused pages in ram bulk stage

- Inefficient data processing in ram bulk stage
  - Unused page can be skipped.
  - Mark all pages as dirty will cause needless data process.



- How to
  - Using a dirty page bitmap which just contains the used pages.
  - Start the log dirty before VM running.

Delay the non-emergency operations

- Do clean up operation after data transfer completion
  - Delay migration_end.
  - Delay blk_mig_cleanup.

Performance

- Performance of multiple thread (de)compression

Settings: speed limit No, Compress thread: 8, Decompress thread: 2, Compression level: 1, 1Gbps NIC, Guest RAM: 4G

› Idle guest

Zlib	Original way	Multi-thread (de)Compression
total time (msec)	3333	1833 (↓45%)
downtime (msec)	100	27 (↓73%)
transferred ram(kB)	363536	107819 (↓70%)
total ram(kB)	4211524	4211524

› Guest with workload writing random numbers to 1GB area of the memory periodically

Zlib	Original way	Multi-thread (de)Compression
total time (msec)	37369	15989(↓57%)
downtime (msec)	337	173(↓48%)
transferred ram(kB)	4274143	1699824(↓60%)
total ram(kB)	4211524	4211524

Performance

- Performance comparison between multiple thread and XBZRLE
  - Migrating a guest with workload which writes random numbers to memory, LZ4 is used to do the (de)compression.

	Original way	Multi-thread (de)compression	XBZRLE	Multi-thread (de)compression & XBZRLE
total time (msec)	26746	14490	17590	13522
downtime (msec)	35	64	185	167
transferred ram(kB)	3354024	1784685	2131286	1605739
total ram(kB)	8405576	8405576	8405576	8405576

Performance

- Performance for skipping unused pages in the ram bulk stage

	Before optimization	After optimization
Total time(ms)	1386	483(↓65%)
Transferred ram(KB)	446542	428300
Total ram (KB)	8405576	8405576

Idle guest, 10Gbps NIC

Performance

- Performance for delay the clean up operation

	Before optimization	After optimization
Down time(ms)	38	6(↓84%)
Total ram (KB)	8405576	8405576

Test is based on QEMU 2.4.0 + Linux kernel 4.2-rc6, idle guest. Set max downtime 0.01S

Work in progress

- Improve the performance of multi-thread (de)compression in 10G network environment.
  - With the multi-thread (de)compression on, the performance is worse.
- Improve the performance of the hardware compression accelerator.
  - Using the asynchronous mode instead of the synchronous mode.
- Using AVX instruction to accelerate zero page checking.
- User space network stack
  - Live migration based on DPDK & mTCP
- Live migration performance optimization for the 40Gbps network

Q&A?

Thank You