Software-based Live Migration for Containerized RDMA

Xiaoyu Li [1, 2, 3], Ran Shu [3], Yongqiang Xiong [3], Fengyuan Ren [1, 2]

[1] Tsinghua University, [2] BNRist, [3] Microsoft Research

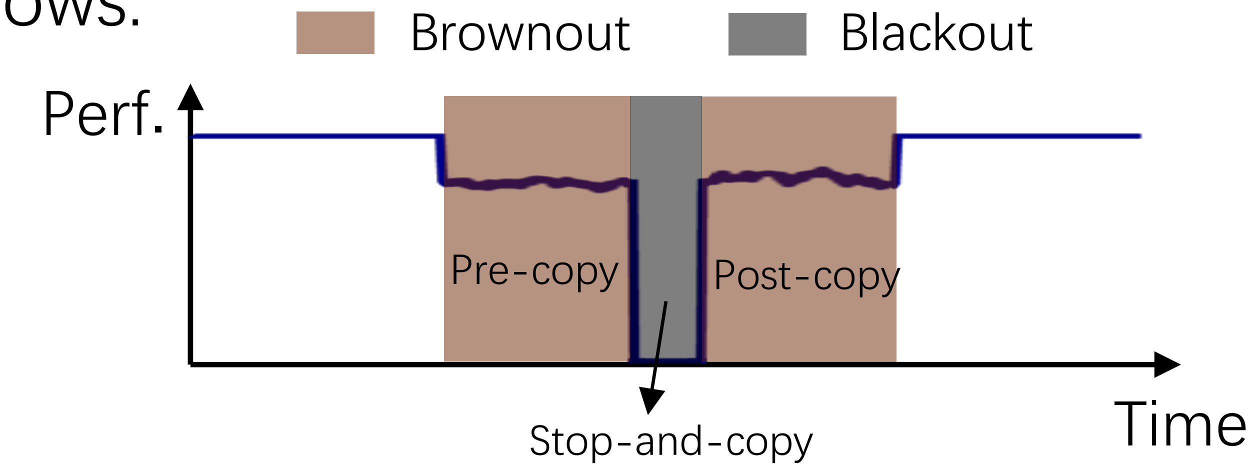
APNet 2024

Background – Container Live Migration

- Container have become the de facto choice of modern data centers

Microsoft Azure aws Google Cloud Alibaba Cloud

- Container live migration:
  - Moving containers from one host to another, without interrupting the services
  - Critical to server upgrades, data center maintenance, and load balancing
- Workflows:



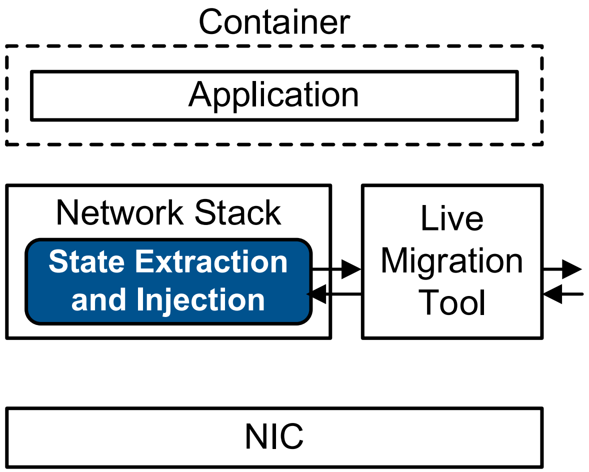
Goals:

Minimize blackout time

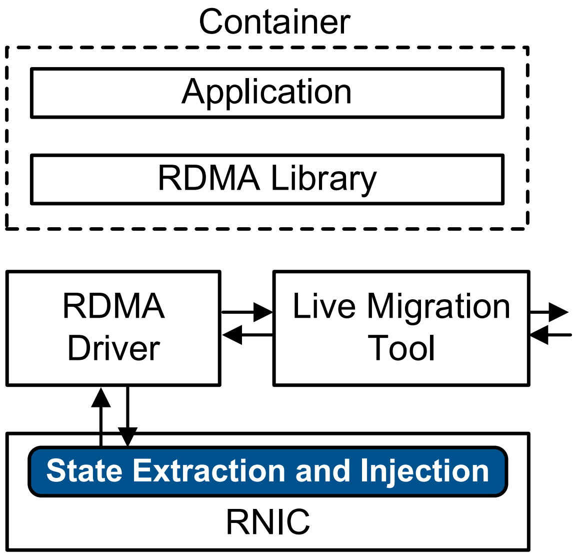
Reduce impacts of brownout

Background – Container Live Migration with RDMA

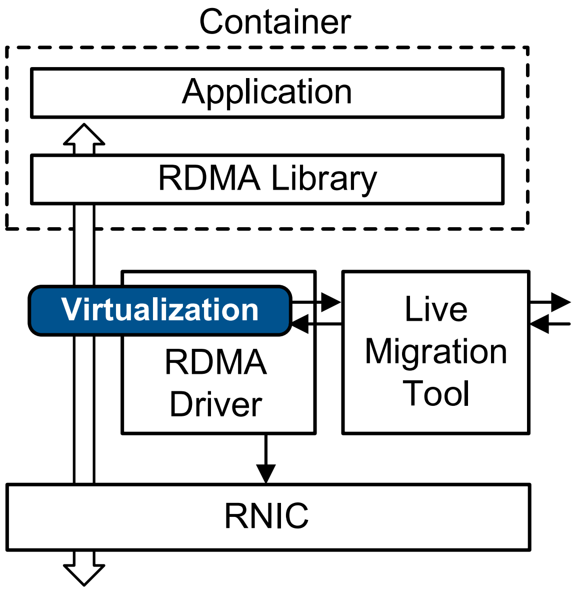
- Wide deployment of RDMA in data centers, RDMA containerization has become focus
- RDMA live migration for containers is impossible today
  - Reason: Most states managed by RNICs, no interfaces for migration
- Existing solutions for TCP/RDMA live migration, and our work:



TCP Live Migration
(from Linux 3.5)



MigrOS [ATC'21]



MigrRDMA (Our work)

Background – Container Live Migration with RDMA

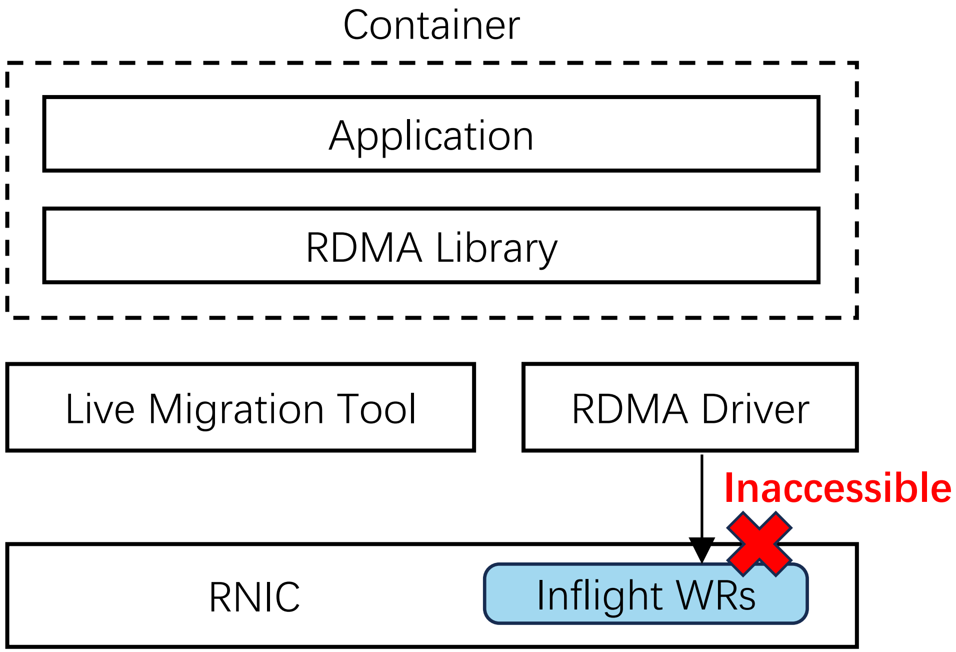
- Challenge #1: Setting up communication using pre-copy approach
    - Slow RDMA communication setup (several ms per connection) ➔ Need to adopt pre-copy approach
    - Adopting a pre-copy approach is challenging:
        - Establishing RDMA needs registering application memory
        - Containers to be migrated are still running on the migration source
        - Memory has not been allocated on the migration destination yet

Background – Container Live Migration with RDMA

- Challenge #2: Lightweight virtualization
  - Some states are managed by RNICs ➔ Need to virtualize them for transparency
  - These states are used in the data path
  - Hard to add the virtualization layer in the data path
    - RDMA’s data path bypasses the kernel
  - Translation during data transmission is prone to performance declines
    - Each operation takes only ~100 CPU cycles

Background – Container Live Migration with RDMA

- Challenge #3: Consistency of inflight work requests
  - RNIC will continue operating the messages even though containers are frozen
  - Stopping all the RDMA connections is slow
  - The states of inflight work requests (WRs) cannot be accessed by the software
  - The migrated container is unaware of the one-sided verbs issued by the partners



Design – Partial Restore for RDMA Pre-setup

- RDMA pre-copy
  - Pre-copying RDMA connection states together with container memory pre-copy
  - Pre-copy the basic container states originally copied in stop-and-copy
- Container partial-restore during pre-copy phase
  - Start restoring the container with basic states
  - Only restore the minimal states for setup RDMA related memory
  - Setup RDMA connections during partial-restore
    - Memory pre-copy like approach to ensure RDMA connections convergence
- Container full-restore during stop-and-copy phase
  - Restore remaining container states

Design – Lightweight Virtualization

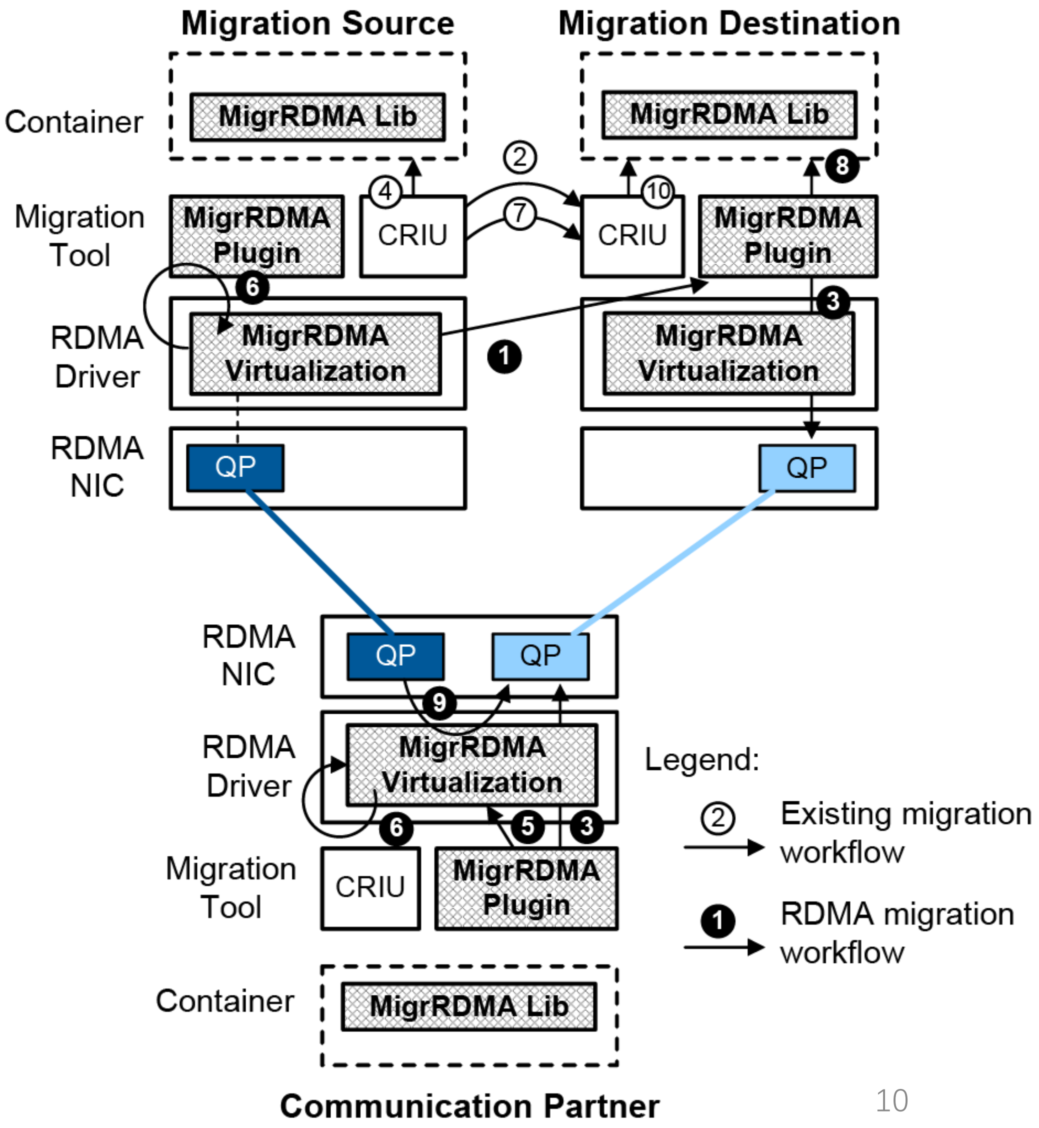
- Three categories of states managed by RNICs and used in the data path:
    - Maintained in the metadata of library, application uses a handle to access the metadata (local QPN)
        - Update the physical ones after migration by the live migration tools
    - Applications use the values directly (local access key)
        - Maintain translation table + memory mapping
        - Store the translation table closely together (as arrays)
    - Exchanged out-of-band, and the exchange is unknown to the RDMA Library or driver (remote access key, and remote QPN for UD QP)
        - Cache remote mapping table
        - Fetch from the remote side for the first time
        - Invalidated after migration

Design – Inflight WR Consistency

- Wait before state transferring:
  - Stop → Wait for completion of inflight WRs → Transferring states
  - A “fake” CQ to store the CQEs consumed by MigrRDMA
- Partner’s QP suspension in the software:
  - Partners may continue sending messages to the migrated containers
  - The migrated containers are unaware of the one-sided verbs from the partners

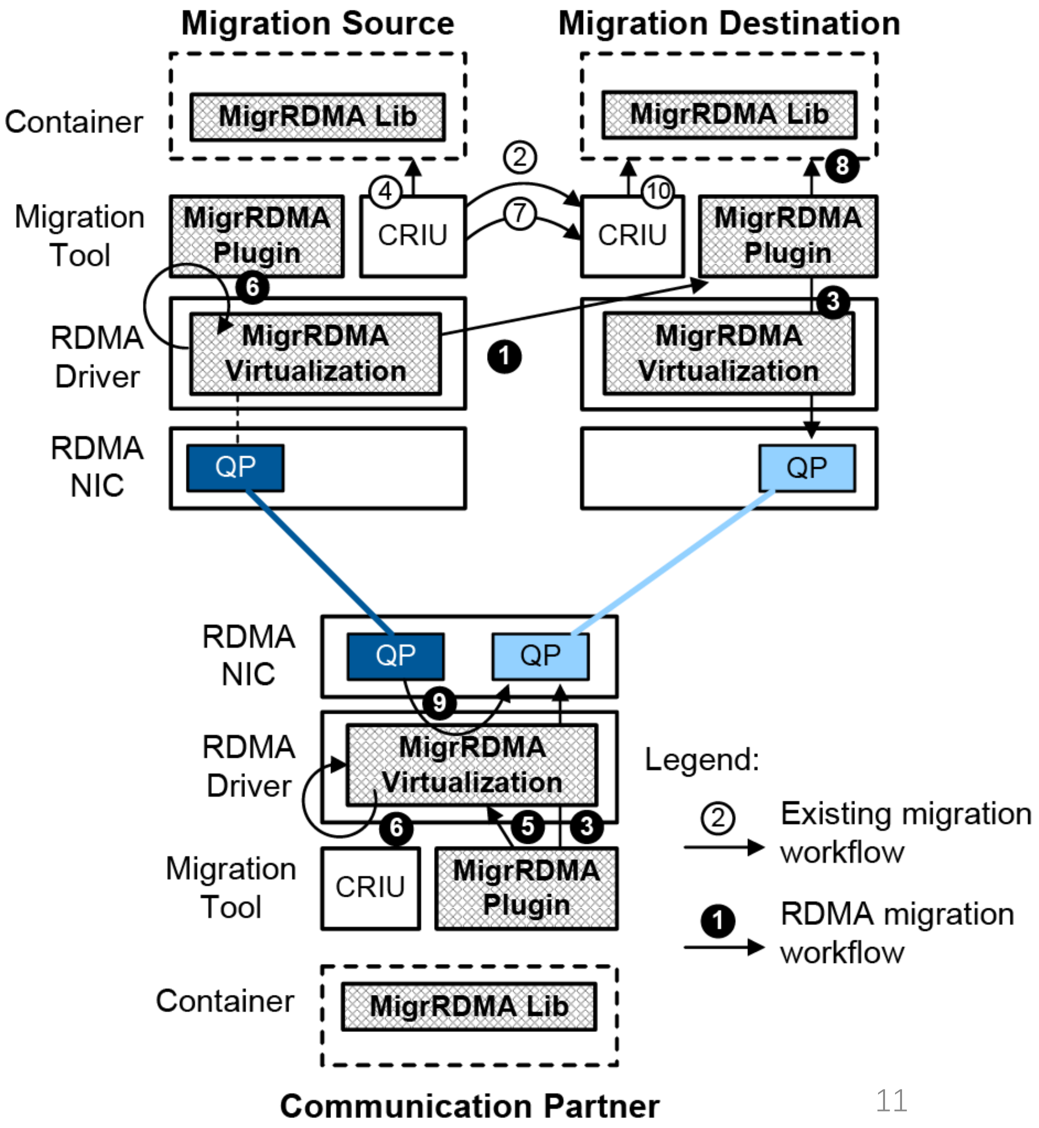
Overall Design

- MigrRDMA architecture:
  - MigrRDMA Virtualization
    - Manage the translation table
    - Maintain the RDMA information for migration tools to checkpoint
  - MigrRDMA Lib
    - Perform translation
    - Handle inflight WRs
  - MigrRDMA Plugin
    - Checkpoint and pre-copy RDMA information
    - Pre-establish RDMA communication



Overall Design

- MigrRDMA overview:
  - ❶ RDMA information pre-copy
  - ② Pre-copy
  - ❸ Communication pre-setup (during partial restore)
  - ④ Freeze
  - ❺ Partner's QP suspension
  - ❻ Wait for inflight WR's completion
  - ⑦ Copy
  - ❽ Update virtualization
  - ❾ Switch QP
  - ⑩ Restore



Evaluation – Migration Time (ms)

# of QPs	Baseline	MigrRDMA	
	Blackout Time	Blackout Time	Extra Downtime
1	67.5	68.6	1.1
2	69.5	70.5	1.0
4	70.7	71.4	0.7
8	71.9	72.9	1.0
16	72.5	74.3	1.8
32	73.8	79.8	6.0
64	74.6	85.1	10.5
128	77.5	89.6	12.1

* Baseline: Allocate all the necessary memory

- Only adds 0.7 ~ 12.1 ms
- For larger number of QPs, the extra downtime is dominated by wait-before-stop

Evaluation – Virtualization Overhead

Operation	w/o virt	with virt	Extra cycles	overheads
send	123.7	128.3	4.6	3.7%
recv	59.4	64.7	5.3	8.9%
write	125.0	133.3	8.3	6.6%
read	127.3	133.8	6.5	5.1%

- Incurs 3% ~ 9% extra CPU overheads in the data path

Conclusion

- Supporting RDMA live migration readily deployable on commodity RNICs has three challenges:
  - Setting up RDMA connections using pre-copy approach
  - Lightweight virtualization
  - Consistency of inflight work requests
- MigrRDMA is the first software based RDMA live migration solution:
  - Partial-restore during pre-copy
  - Lightweight translation of states used in the data path (three categories)
  - Wait-before-stop
- Evaluations demonstrate low extra migration time and low virtualization overhead

Thanks!
Q & A