Welcome!

ExaNoDe

European Exascale Processor & Memory Node Design

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 671578

Virtualization for HPC
in the ExaNoDe/ExaNeSt projects

ExascaleHPC workshop
HiPEAC conference, Manchester, UK

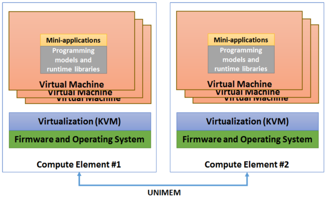
K Pouget, A. Mouzakitis,
R. Dimitrov, A. Rigo, D. Raho

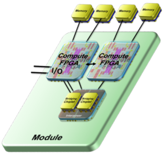
Virtual Open Systems

January 23rd 2018

Introduction

Virtualization in ExaNoDe/ExaNeSt and HPC





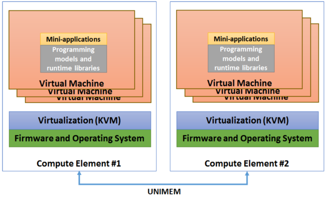
Virtualization layer

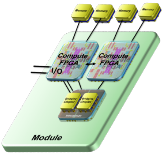
$\Longleftrightarrow$

extra level of hardware resource control

Introduction

Virtualization in ExaNoDe/ExaNeSt and HPC





Virtualization layer

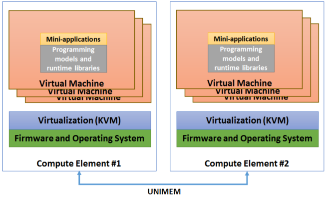
$\Longleftrightarrow$

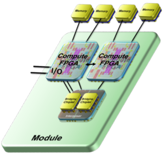
extra level of hardware resource control

- Manageability for admins
  - pause/unload/migrate VMs

Introduction

Virtualization in ExaNoDe/ExaNeSt and HPC





Virtualization layer

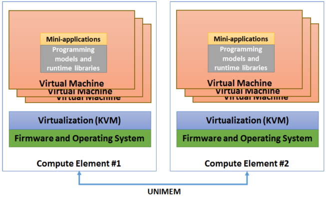
$\Longleftrightarrow$

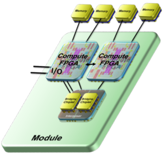
extra level of hardware resource control

- Manageability for admins
  - ◇ pause/unload/migrate VMs
- Flexibility for users
  - ◇ full control of SW eco-system
    - – kernel, libraries, filesystem

Introduction

Virtualization in ExaNoDe/ExaNeSt and HPC





Virtualization layer

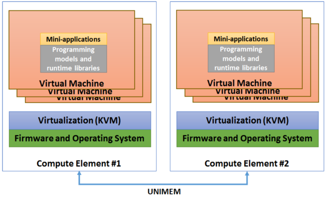
$\Longleftrightarrow$

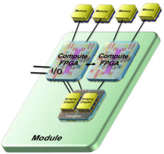
extra level of hardware resource control

- Manageability for admins
  - ◇ pause/unload/migrate VMs
- Flexibility for users
  - ◇ full control of SW eco-system
    - – kernel, libraries, filesystem
- Resiliency to hardware failures
  - ◇ periodic state checkpointing

Introduction

Virtualization in ExaNoDe/ExaNeSt and HPC





Virtualization layer

$\Longleftrightarrow$

extra level of hardware resource control

- Manageability for admins
  - ◇ pause/unload/migrate VMs
- Flexibility for users
  - ◇ full control of SW eco-system
    - – kernel, libraries, filesystem
- Resiliency to hardware failures
  - ◇ periodic state checkpointing
- Performance?
  - ◇ paravirtualization/network optim.

Virtualization for HPC

in the ExaNoDe/ExaNeSt projects

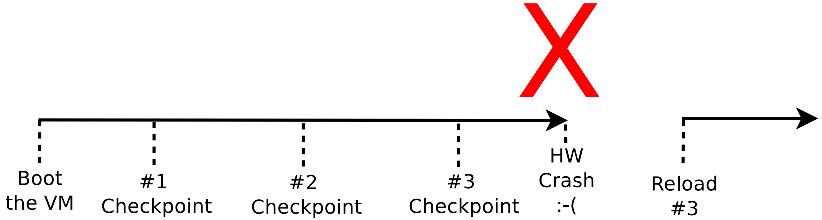
Live and Incremental Checkpointing

Live and Incremental Checkpointing

Definitions and Challenges

Checkpointing

Saving periodically the state of a VM to file, to restore it later, maybe on another machine.

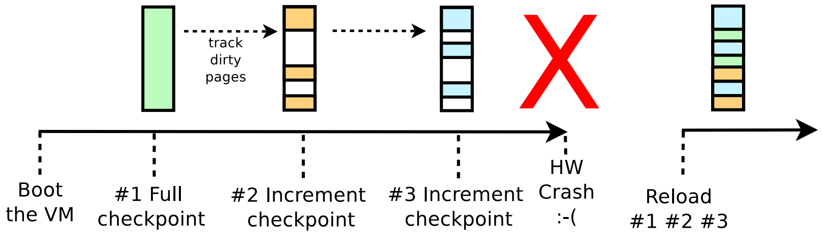


Live and Incremental Checkpointing

Definitions and Challenges

Incremental checkpoint

A full checkpoint copies the whole memory to the disk.
An incremental one copies only the modified (dirty) pages.
✓ less guest downtime and less disk occupation

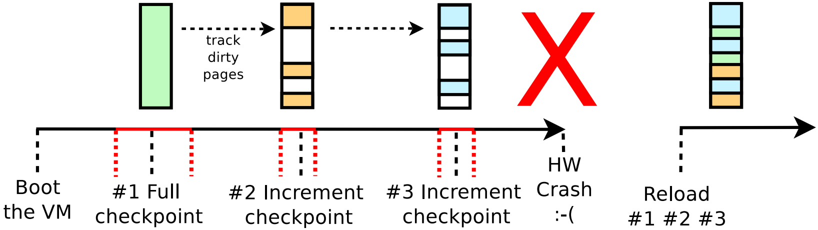


Live and Incremental Checkpointing

Definitions and Challenges

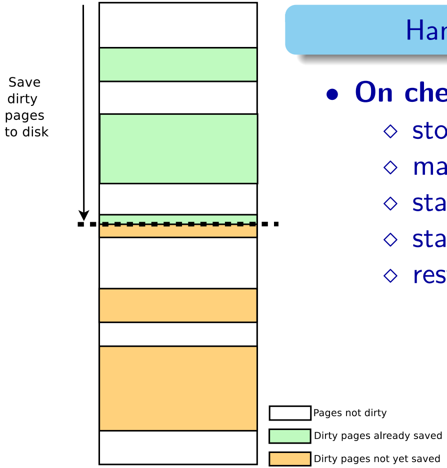
Live checkpointing

Saving the VM memory to disk takes time. During a live checkpoint, the VM is running while the memory is being copied.
✓ less guest downtime



Live and Incremental Checkpointing

Going live and incremental: Page Fault Handling

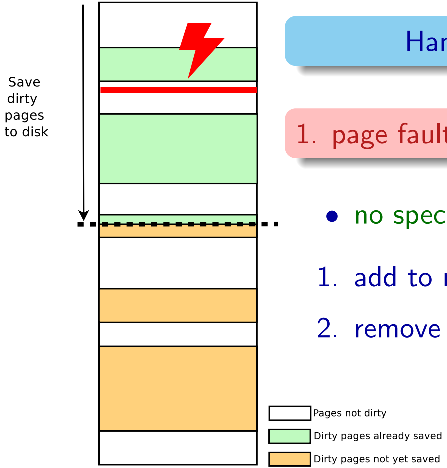


Handling of page write faults

- **On checkpoint request:**
  - ◇ stop the VM
  - ◇ mark all the pages as write-protected
  - ◇ start sequential page saving thread
  - ◇ start fault-handler thread
  - ◇ restart the VM

Live and Incremental Checkpointing

Going live <u>and</u> incremental: Page Fault Handling



Handling of page write faults

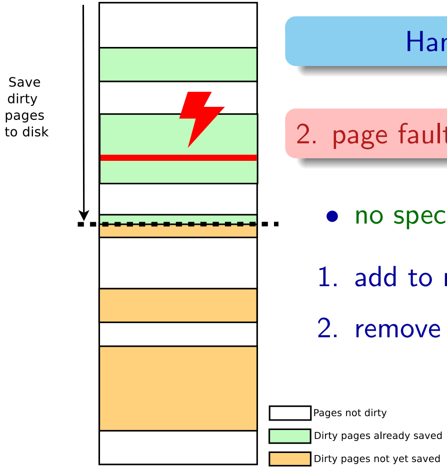
1. page fault on page not dirty

- no special care

1. add to next checkpoint dirty queue
2. remove the page write-protection

Live and Incremental Checkpointing

Going live and incremental: Page Fault Handling



Handling of page write faults

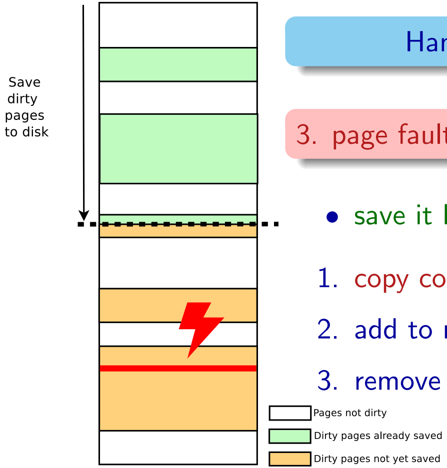
2. page fault on already saved dirty page

- no special care

1. add to next checkpoint dirty queue
2. remove the page write-protection

Live and Incremental Checkpointing

Going live and incremental: Page Fault Handling



Handling of page write faults

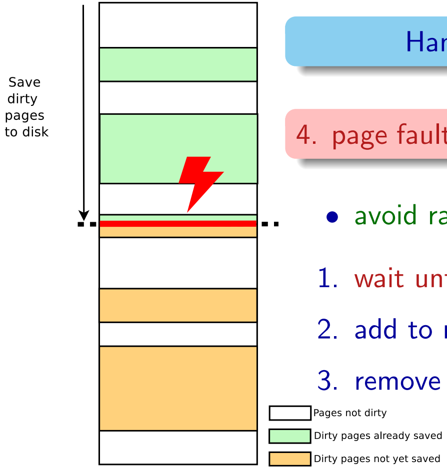
3. page fault on dirty page *not* saved to disk

- save it before it's overwritten!

1. copy content to shadow memory
2. add to next checkpoint dirty queue
3. remove the page write-protection

Live and Incremental Checkpointing

Going live and incremental: Page Fault Handling



Handling of page write faults

4. page fault the dirty page *being* saved to disk

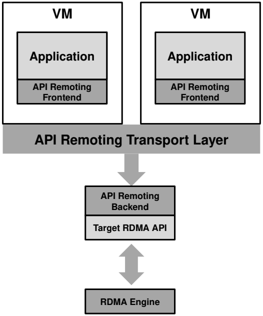
- avoid race condition, block until saved

1. wait until page marked as saved
2. add to next checkpoint dirty queue
3. remove the page write-protection

Unimem RDMA Para-Virtualization

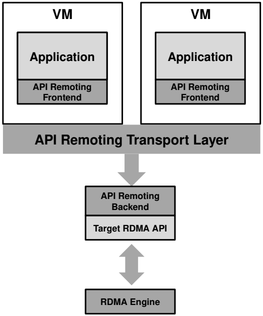
API Remoting Layers

API Remoting para-virtualization allows calling host libraries from inside virtual machines.
e.g. : for accessing hardware accelerators.



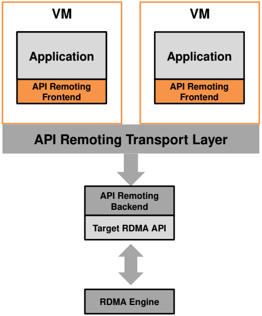
API Remoting Layers

- Frontend
  - ◇ Shared library stub in the guest
  - ◇ Implementing the remoted API
  - ◇ Forwarding the requests to the host
- Transport layer
  - ◇ Host-VM shared memory + `virtio`
  - ◇ Zero-copy accesses to DMA buffers
- Backend
- RDMA library



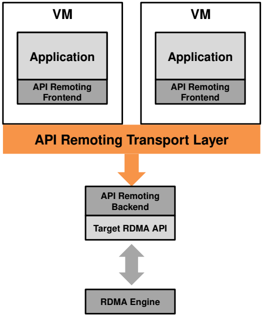
API Remoting Layers

- Frontend
  - ◇ Shared library stub in the guest
  - ◇ Implementing the remoted API
  - ◇ Forwarding the requests to the host
- Transport layer
  - ◇ Host-VM shared memory + `virtio`
  - ◇ Zero-copy accesses to DMA buffers
- Backend
- RDMA library



API Remoting Layers

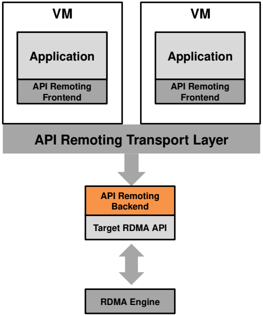
- Frontend
  - ◇ Shared library stub in the guest
  - ◇ Implementing the remoted API
  - ◇ Forwarding the requests to the host
- Transport layer
  - ◇ Host-VM shared memory + `virtio`
  - ◇ Zero-copy accesses to DMA buffers
- Backend
- RDMA library



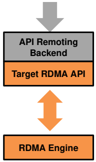
The Backend

- Host process linked to the remoted library
  - ◇ handles the guest requests
  - ◇ sends back return values/errors

- Listening for new VMs and applications
  - ◇ supports multiple VMs
  - ◇ supports multiple apps per VM
  - ◇ spawns one thread per guest application
  - ◇ each thread has a private . . .
    - – shared memory space with frontend
    - – file descriptor to the zero-copy framework in the kernel



The RDMA Library



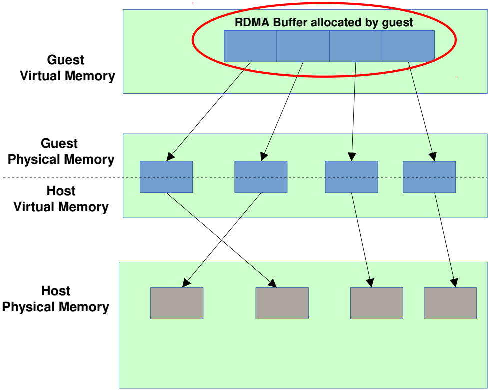
- Custom API from FORTH (cDMA)
- Support synchronous and asynchronous transfers
  - Polling mode
  - Interrupt mode
- Transactions programmed using a user-space driver
  - DMA buffer allocation managed in the kernel

RDMA Buffer Allocation

On a DMA buffer allocation call from the application

in the frontend (guest)

1. allocate the space with `mmap`

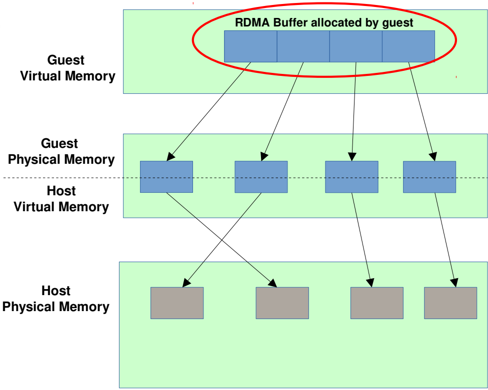


RDMA Buffer Allocation

On a DMA buffer allocation call from the application

in the frontend (guest)

1. allocate the space with `mmap`
2. send the Virtual Address (VA) to the backend



 January 23rd 2018 ExascaleHPC workshop Virtual Open Systems

RDMA Buffer Allocation

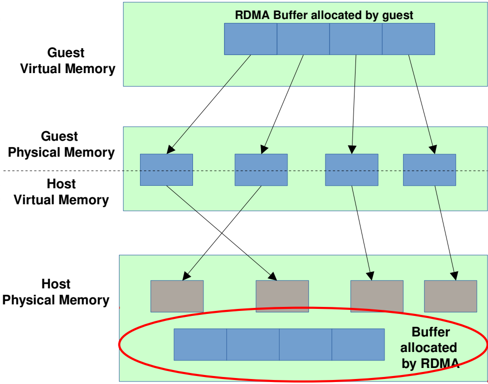
On a DMA buffer allocation call from the application

in the frontend (guest)

1. allocate the space with `mmap`
2. send the VA to the backend

in the backend

3. allocate the DMA buffer



RDMA Buffer Allocation

On a DMA buffer allocation call from the application

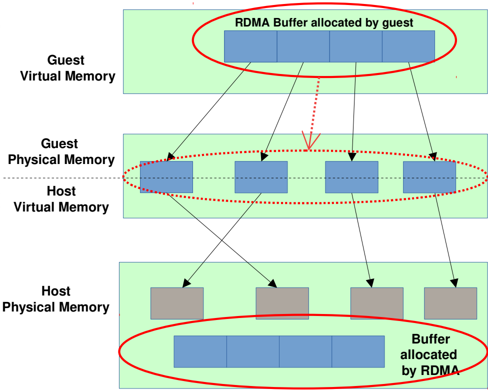
in the frontend (guest)

1. allocate the space with `mmap`
2. send the VA to the backend

in the backend

3. allocate the DMA buffer
4. pass to the kernel :
   - the DMA buffer's VA
   - the guest buffer's VA

   ⇒ physical address looked up



RDMA Buffer Allocation

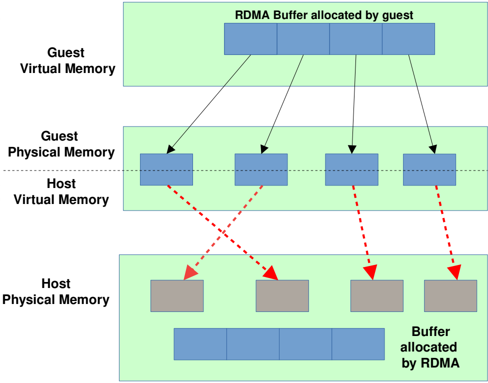
On a DMA buffer allocation call from the application

in the frontend (guest)

1. allocate the space with `mmap`
2. send the VA to the backend

in the backend

3. allocate the DMA buffer
4. pass the VAs to the kernel
5. remove guest buffer pages from QEMU address space



RDMA Buffer Allocation

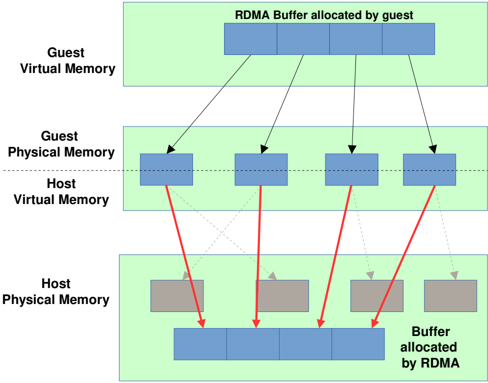
On a DMA buffer allocation call from the application

in the frontend (guest)

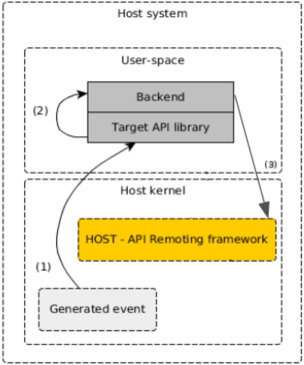
1. allocate the space with `mmap`
2. send the VA to the backend

in the backend

3. allocate the DMA buffer
4. pass the VAs to the kernel
5. remove guest buffer pages from QEMU address space
6. insert in-place the pages of the DMA buffer

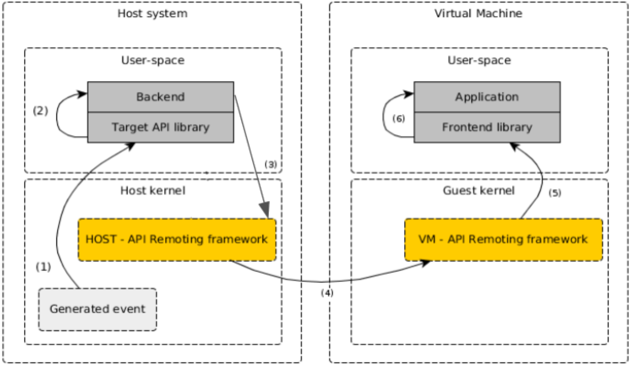


Completion-event forwarding



1. the RDMA generates an event
2. and dispatches it to the backend;
3. the event is forwarded to the VM
   ... through the host kernel;

Completion-event forwarding

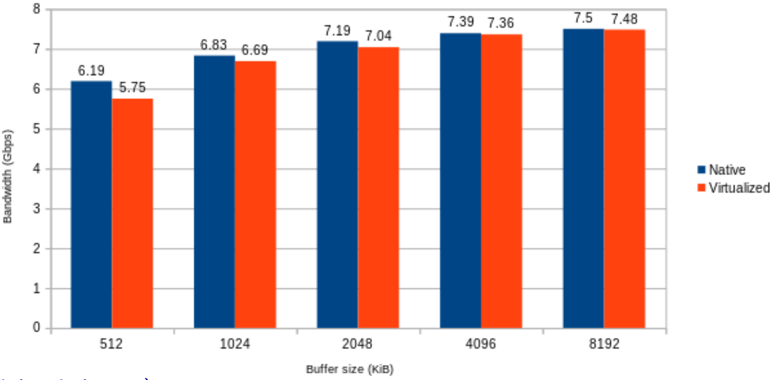


1. the RDMA generates an event
2. and dispatches it to the backend;
3. the event is forwarded to the VM … through the host kernel;
4. the guest kernel receives it (via a `virtio` message)
5. and signals the frontend library;
6. the frontend runs the callback

Experimental Results

Transfer rate for a single transfer

- Maximum of 7% overhead (with 512KiB buffers)

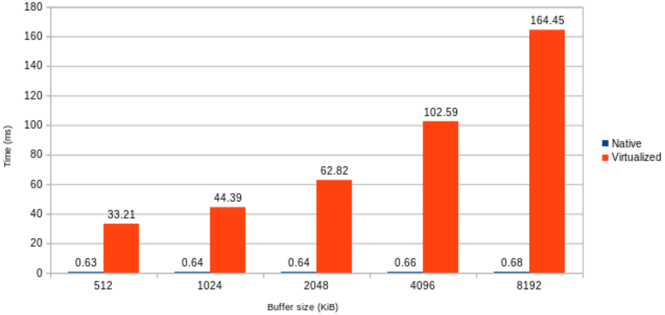


(higher is better)

Experimental Results

DMA buffer allocation

- Tangible overhead only at the allocation time



(lower is better)

Conclusion

UNIMEM RDMA Para-Virtualization

Bigger picture in ExaNoDe/ExaNeSt

High-performance multi-host virtualization layer

API Remoting | Resiliency | Networking

UNIMEM RDMA Para-Virtualization

Bigger picture in ExaNoDe/ExaNeSt

High-performance multi-host virtualization layer

API Remoting

- UNIMEM RDMA, mailbox, atomics
- MPI, OpenCL

Resiliency

Networking

UNIMEM RDMA Para-Virtualization

Bigger picture in ExaNoDe/ExaNeSt

High-performance multi-host virtualization layer

API Remoting

- UNIMEM RDMA, mailbox, atomics
- MPI, OpenCL

Resiliency

- multi-host/VM checkpointing
- live migration

Networking

UNIMEM RDMA Para-Virtualization

Bigger picture in ExaNoDe/ExaNeSt

High-performance multi-host virtualization layer

API Remoting

- UNIMEM RDMA, mailbox, atomics
- MPI, OpenCL

Resiliency

- multi-host/VM checkpointing
- live migration

Networking

Optimize communications

- between same-host VM
- multi-host by UNIMEM

UNIMEM RDMA Para-Virtualization

Bigger picture in ExaNoDe/ExaNeSt

High-performance multi-host virtualization layer

API Remoting

- UNIMEM RDMA, mailbox, atomics
- MPI, OpenCL

Resiliency

- multi-host/VM checkpointing
- live migration

Networking

Optimize communications

- between same-host VM
- multi-host by UNIMEM

Future work

Assemble all of that, and get it running in the prototypes!

Thank you!

European Exascale Processor & Memory Node Design