RDMA CONTAINERS UPDATE

Haggai Eran, Liran Liss, Parav Pandit
Mellanox Technologies Hewlett Packard Enterprise

April 5th, 2016
CONTAINERS

- Isolation
- Resource control
- Lightweight virtualization
BENEFIT OF CONTAINERS

Image credit: Apache Mesos at twitter Texas Linuxfest 2014
RDMA CONTAINERS

Container A
Process
Kernel

Container B
Process
AGENDA

- Containers 101
- RDMA network namespace support
  - InfiniBand
  - RoCE
- RDMA cgroup
- Future work
CONTAINERS 101

- A server-virtualization technology for running multiple isolated user-space instances
- Each instance
  - Has the look and feel of running over a dedicated server
  - Cannot impact the activity of other instances
- Containers and Virtual Machines (VMs) provide virtualization at different levels
EXAMPLE: DOCKER

- **Open platform to build, ship, and run distributed apps**
  - Based on Linux container technology

- **Main promise**
  - Easily package an application and its dependencies
    - Regardless of the language, tool chain, and distribution
    - Layered images
    - Large application repository
      - Basis for further specialization
  - Deploy on any Server
    - Regardless of OS distribution
    - Regardless of underlying architecture
  - Lightweight runtime
    - Rapid scale-up/down of services

---

**Docker Image**

- **App**
- **Math libs**
- **Lustre**
- **OpenMPI**
- **OFED**
- **RHEL**

**App layers**

**Base layers**
LINUX CONTAINERS = NAMESPACES + CGROUPS

- **Namespaces**
  - Provide the illusion of running in isolation
  - Implemented for multiple OS subsystems

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid</td>
<td>Process IDs</td>
</tr>
<tr>
<td>net</td>
<td>Network interfaces, routing tables, and netfilter</td>
</tr>
<tr>
<td>ipc</td>
<td>Semaphores, shared memory, and message queues</td>
</tr>
<tr>
<td>mnt</td>
<td>Root and file-system mounts</td>
</tr>
<tr>
<td>uts</td>
<td>Host name</td>
</tr>
<tr>
<td>uid</td>
<td>User IDs</td>
</tr>
</tbody>
</table>

- **cgroups**
  - Restrict resource utilization
  - Controllers for multiple resource types

**Namespace examples**

<table>
<thead>
<tr>
<th>Controller</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blkio</td>
<td>Access to block devices</td>
</tr>
<tr>
<td>cpu</td>
<td>CPU time</td>
</tr>
<tr>
<td>cpuset</td>
<td>CPU cores</td>
</tr>
<tr>
<td>devices</td>
<td>Device access</td>
</tr>
<tr>
<td>memory</td>
<td>Memory usage</td>
</tr>
<tr>
<td>net_cls</td>
<td>Packet classification</td>
</tr>
<tr>
<td>net_prio</td>
<td>Packet priority</td>
</tr>
<tr>
<td>RDMA</td>
<td>RDMA resources. Explained later.</td>
</tr>
</tbody>
</table>
**CONTAINER IP NETWORKING**

### Common models
- Host
- Physical interface / VLAN / macvlan / ipvlan
  - Container has global IP
- Bridge
  - Container has global IP
- Pod (e.g., GCE)
  - Multi-container scheduling unit
  - Global IP per POD
- NAT (e.g., Docker)
- Tunneling (VXLAN with docker multi-host)

### Building blocks
- Network namespaces
  - Interfaces, IP tables, netfilter
- Virtual networking
  - bridge, ovs, NAT
  - macvlan, vlan, veth
RDMA NETWORK NAME SPACE SUPPORT
RDMA ISOLATION DESIGN GOALS

- **Simplicity and efficiency**
  - Containers share the same RDMA device instance
  - Leverage existing isolation infrastructure
    - Network namespaces and cgroups

- **Focus on application APIs**
  - Verbs / RDMACM
  - Exclude management and low-level APIs (e.g., umad, ucm)
    - Deny access using device controller
  - Exclude kernel ULPs (e.g., iSER, SRP)
    - Not directly exposed to applications
    - Controlled by other means (blk_io)
  - Subject for future work
RDMA CM IN NET. NAMESPACE

- Have per-namespace RDMA CM port spaces
- Upon creation of an rdma_cm_id associate it with the creating process’s namespace
- De-multiplex requests
MAD SERVICEID RESOLUTION

ib_cma

Lookup NS

ib_cm

Host NS

NO

YES

Is IP CM ServiceID?

CM MADs

ib_core

Lookup NS

Get cm_id NS

Match existing cm_id

Find listener id in NS

Get netdev NS

Match netdev

YES

NO

Is solicited?
**Initial proposal**

- No need for support at the verbs level
  - No user-chosen well-known QPs, R_Keys.
- RDMA CM support for network namespace
  - Select net namespace for incoming RDMA CM requests based on:
    - IB device
    - Physical port
    - IP address
    - P_Key

**Rejected because demux doesn’t look at the GID**
- Modified proposal was accepted into kernel 4.4

- De-mux incoming requests according to:
  - IB device
  - Physical port
  - GID (treat as a layer two address)
  - P_Key
    - Analogous to macvlan

- If above properties don’t provide a unique namespace
  - (e.g. when creating multiple interfaces with `ip link add`)
  - Use IP address to de-mux
    - Analogous to ipvlan
ROCE NET. NAMESPACENT SUPPORT

- De-mux RDMA CM requests
  - Use new GID table
- verbs: Route and neighbor lookup
- Filter supported GIDs
Kernel v4.4 added code using new RoCE GID table to the core

Each GID is associated with its netdevice

The netdevice is associated with its net namespace

Code automatically adds new GIDs for

- New IP addresses
- macvlan / vlan devices
- bonding devices
De-muxing RDMA CM requests becomes as simple as finding the matching GID for an incoming request

- RDMA device
- Physical port
- MAC
- VLAN
- IP address
- RoCE version
VERBS ROUTE AND NEIGH. LOOKUP

- RoCE does route and neighbor lookup
  - Finding source GID, destination MAC and VLAN given a destination IP.
  - `ibvmodify_qp()` (RC change to RTR)
  - `ibv_create_ah()` (UD)

- Do the route and neighbor lookup in the namespace of the calling process
FILTER SUPPORTED GIDS

- A RoCE devices’ GID table includes GIDs from all namespaces
- Each process has to see only its own namespace’s subset
  - Filter sysfs GIDs
  - `ibv_query_gid()` is implemented by querying sysfs
- Enforce only valid GIDs are passed to `ibv_modify_qp` and `ibv_create_ah`
RDMA CGROUP
RDMA CGROUP CONTROLLER

- Started as device cgroup extension, but community asked for dedicated cgroup.
- It is an individual rdma cgroup now
- **Governs application resource utilization**
  - Per device resource configuration
  - For a process or a group of processes
  - Supports hierarchical resource accounting, allows running nested containers
  - Ensures applications cannot take away all the resources, allowing kernel consumers to make use of it.
- **Controlled resources**
  - Opened HCA contexts
  - AHs, CQs, PDs, QPs, SRQs, MRs, MWs, Flows
- **New RDMA cgroup resources can be defined and charged without kernel upgrade by the IB stack**
RDMA CGROUP IMPLEMENTATION

![Diagram showing cgroup hierarchy with root and child groups labeled A, B, C, D, E.]](image-url)
MIGRATING PROCESSES

- Process charges resource to its cgroup

- **Migrate to another cgroup**
  - Might even fork before migrating (shared resources)
  - Not a common use case so pick the simpler solution
  - Resource charging always occurs to current owner cgroup
  - After migration resources are changed to newer cgroup
  - Resources created before migration continues with older cgroup

- **Release resource**
  - Resources are uncharged from their original cgroup
  - Each resource (*ib_uobject*) points to its cgroup
RDMA CGROUP EXAMPLES

- **Raw file system level examples**
  - Creating/Deleting rdma cgroup
    ```
    # cgcreate –g rdma:c1
    # cgdelete –g rdma:c2
    ```
  - Configuring resource limits
    ```
    # echo mlx4_0 qp=10 mr=8 > /sys/fs/cgroup/rdma/c1/rdma.max
    ```
  - Query resource limits
    ```
    # cat /sys/fs/cgroup/rdma/c1/rdma.max
    ```
    Output:
    mlx4_0 uctx=max ah=max mr=7 cq=max srq=max qp=9 flow=max
    ocrdma0 uctx=max ah=max mr=max cq=max srq=max qp=max flow=max
  - Delete resource limits
    ```
    # echo mlx4_0 qp=max mr=max > /sys/fs/cgroup/rdma/c1/rdma.max
    ```
  - Query resource usage
    ```
    # cat /sys/fs/cgroup/rdma/c1/rdma.current
    ```
    Output:
    mlx4_0 uctx=max ah=max mr=8 cq=max srq=max qp=10 flow=max
    ocrdma0 uctx=max ah=max mr=max cq=max srq=max qp=max flow=max
From alf87459e12055b287adefcb79cc8ef9b9b3dc6c Mon Sep 17 00:00:00 2001
From: Parav Pandit <parav.k.pandit@hpe.com>
Date: Sun, 20 Mar 2016 18:15:31 +0530
Subject: [PATCH] cgroup/rdma: Added support for Rdma cgroup.

This patchset added support to configure rdma controller limits
of each container instance.
RDMA controller limits are configured at instance launch time using
--rdma-limit option.

It allows updating limits using Docker Remote API JSON extension
using "RdmaLimit" key.

Parav Pandit (1):
cgroup/rdma: Added support for RDMA cgroup controller.

daemon/container_operations_unix.go | 1 +
daemon/daemon_unix.go | 5 +++
daemon/execdriver/driver_unix.go | 2 +
pkg/sysinfo/sysinfo.go | 6 +++
pkg/sysinfo/sysinfo_linux.go | 16 ++++++ runconfig/opts/parse.go | 2 +
.../engine-api/types/container/host_config.go | 1 +
.../runc/libcontainer/cgroups/fs/apply_raw.go | 1 +
.../runc/libcontainer/cgroups/fs/rdma.go | 51 ++++++++++++++++++++++++++
.../runc/libcontainer/cgroups/stats.go | 5 ++
.../runc/libcontainer/cgroups/configs/cgroup_unix.go | 3 ++
11 files changed, 93 insertions(+)
create mode 100644 vendor/src/github.com/opencontainers/runc/libcontainer/cgroups/fs/rdma.go
RDMA CGROUP DOCKER EXAMPLES

- Resource configuration at starting Docker container:
  - `docker run --net=host --rdma_limit="mlx4_0 cq=10 qp=100 mr=4" -i -t /bin/bash`

- Docker Remote API
  
  Example request:
  ```
  POST /containers/(id)/update HTTP/1.1
  Content-Type: application/json
  {
    "RdmaLimit": "mlx4_1 flow=100 qp=200",
  }
  ```

  Example response:
  ```
  HTTP/1.1 200 OK
  Content-Type: application/json
  {
    "Warnings": []
  }
  ```

- Inside Docker instance
  ```
  # cat /sys/fs/cgroup/rdma/rdma.max
  Output:
  mlx4_1 uctx=max ah=max mr=7 cq=max srq=max qp=200 flow=100
  ```
STATUS

- InfiniBand RDMA CM support in v4.4
- Working on RoCE net namespace support
- RDMA cgroup patches submitted
- RDMA cgroup Docker patches are ready, will be submitted once kernel patches are accepted

Future work
- InfiniBand: limit P-Key usage in verbs applications
  - Perhaps extend the RDMA cgroup or use the new SELinux patches
- QoS: limit container’s bandwidth usage, SL, or VLAN priority
- Raw Ethernet support
CONCLUSION

- RDMA container technology provides HPC applications access to high-performance networking in a secure and isolated manner

- RDMA cgroups allow fine grained control over RDMA resource allowing better utilization of available hardware

Thank you
12th ANNUAL WORKSHOP 2016

THANK YOU

Haggai Eran, Liran Liss, Parav Pandit
Mellanox Technologies Hewlett Packard Enterprise
MOTIVATIONAL EXAMPLE

Net NS: 1
  cpu: 10%
  Cgroup: 1
  QPs: 10
  CQs: 10

App A
  listen rdma_id:
  TCP port-space 2000

ib_0
  0x8001
  10.2.0.1
ib_1
  0x8001
  10.2.0.2

Net NS: 2
  cpu: 20%
  Cgroup: 2
  QPs: 50
  CQs: 50

App B
  listen rdma_id:
  TCP port-space 2000

ib_2
  0x8002
  10.3.0.1

Net NS: 3
  cpu: 30%
  Cgroup: 3
  QPs: 100
  CQs: 100

App C

ib_0.100
  10.4.0.1
ib_0.101
  10.5.0.1

Linux

eth0

IB core

IB HCA

RoCE HCA

eth0
  11.1.0.1