Enabling HPC workloads on Cloud Infrastructure using Kubernetes Container Orchestration Mechanisms

SC19 Workshop: CANOPIE HPC

Angel Beltre, Pankaj Saha, and Madhusudhan Govindaraju
Binghamton University Big Data and Cloud Laboratory

Andrew Younge and Ryan Grant
Sandia National Laboratories
Motivation

- Containers are changing the software ecosystem for application deployment
- Need to leverage benefits of containers in HPC
  - Easy to manage
  - Software isolation
  - Portability and Reproducibility
- Containers are widely used for Microservices in industry
  - HPC requirements are different than Microservices
- Container orchestration tools are now mainstream
- **Study Opportunities for HPC, Containers, and Container orchestration frameworks**
  - Performance, Usability, and Constraints
Setup: Kubernetes Components

- Kube-apiserver
- Etcd
- Kube-scheduler
- Kube-controller-manager
Setup: Kubernetes Network Instrumentation

- **Traditional Network**
  - Within a node containers have a unique virtual network interface $veth\{0,\ldots,N\}$
  - All containers are connected to the same docker0 bridge within a pod

- **Overlay Network**
  - The combination of interfaces, bridges, and routing rules
  - Consolidation of the network for a single host on a Kubernetes Pod over multiple containers
Kubernetes is pluggable:
- Kubernetes provides a unique way of accessing host nodes’ hardware devices through vendor specific system pods.
  - Physical Function (PF) (SR-IOV).
  - Virtual Function (VF) - Virtual PCIe device.
An overlay network was set up using Flannel to enable inter-Pod communication across nodes.

A device Pod was created to allow device discovery from a container within a Pod.
Evaluation: Latency - AlltoAllv

- Kubernetes has an overhead of 4x than Docker Swarm.
- Docker Swarm best performance improvements occurs between 8KB and 1MB message sizes.
- High performance interconnect reduces overhead for all message sizes to an average deviation of 1%.
- Docker Swarm is wire data rate limited.

OSU AlltoAllv

<table>
<thead>
<tr>
<th>Message Size (Bytes)</th>
<th>TCP</th>
<th>RDMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>66.77%</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15.95%</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32768</td>
<td></td>
<td></td>
</tr>
<tr>
<td>65536</td>
<td></td>
<td></td>
</tr>
<tr>
<td>131072</td>
<td></td>
<td></td>
</tr>
<tr>
<td>262144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>524288</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1048576</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2097152</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TCP

RDMA

OSU AlltoAllv Latency Test

- Bare Metal
- Docker Swarm
- Kubernetes
Evaluation: Bandwidth - Bi-directional

- Kubernetes cannot scale after ~500 MB/s.
- Both Docker Swarm and Kubernetes have relative large overheads in comparison to Bare Metal.

- Both Kubernetes and Docker Swarm achieve near Bare Metal performance levels (i.e., a deviation of about 2%).
Evaluation: Memory - KMI-HASH

- Docker swarm has an overhead of 16.58%.
- Kubernetes has the largest overhead.
- High performance interconnect enables a higher number of queries per second
- Deviation is below 1%

![Bar Chart for TCP and RDMA comparisons]
Docker Swarm yielded the highest overhead.
Only 1 thread over communicating ranks is used.
Both Docker Swarm and Kubernetes yield results below 1%.
- Docker Swarm and Kubernetes yielded substantial overheads in comparison with Bare Metal.

- Both Docker Swarm and Kubernetes yield results below 1%.

**TCP**

-96.17%

-70.14%

**RDMA**
Evaluation: Run Time - SNAP

- Kubernetes present an overhead of 46.64% in comparison to Bare Metal.

- With Infiniband, the overhead was reduced to less than 1% in comparison to Bare Metal for both measurement metrics: Run time and throughput.
Evaluation: Run Time - MiniFE

- Kubernetes presented a large overhead for the 512x512x512 problem size.

- Both Docker Swarm and Kubernetes over IB show an overhead below 1%.

**TCP**

<table>
<thead>
<tr>
<th></th>
<th>Bare Metal</th>
<th>Docker</th>
<th>Swarm</th>
<th>Kubernetes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>150</td>
<td>170</td>
<td>160</td>
<td>160</td>
</tr>
</tbody>
</table>

**RDMA**

<table>
<thead>
<tr>
<th></th>
<th>Bare Metal</th>
<th>Docker</th>
<th>Swarm</th>
<th>Kubernetes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>140</td>
<td>160</td>
<td>160</td>
<td>160</td>
</tr>
</tbody>
</table>
Evaluation: Throughput - MiniFE

- Kubernetes has large performance overheads for both problem sizes.
- Both Docker Swarm and Kubernetes over IB show an overhead below 1%.

![Bar chart showing throughput comparison for MiniFE with Kubernetes, Docker Swarm, and Bare Metal for both TCP and RDMA.]
Evaluation: Throughput - HPCG

- 128 MPI processes, each using about 2.92 GB of memory.
- Kubernetes shows an overhead of 13.15%.
- Docker Swarm achieves Bare Metal performance.

- Kubernetes Overhead was reduced by 2.84% to 10.31% in comparison to Bare Metal.
- Kubernetes shows the inability to scale over TCP/IP for a small grid (P=4 x Q=32) and using about 80% of the clusters total memory.

- With Infiniband, the overhead was reduced to less than 1% in comparison to Bare Metal.

- Docker Swarm was able to outperform.

---

**Throughput - HPL - Evaluation**

![Throughput Graph](image)
We evaluated a diverse set of applications over TCP/IP and RDMA.

We identified that the performance difference is more substantial for memory and network intensive applications.

For all container-based setups, high performance transport expectedly enabled higher performance.

Future work
- Explore scalability.
- Explore network stack overhead via profiling tools to better understand its performance.
- Study other container technologies by enabling different container runtime interfaces (CRI).