Spyre: A Resource Management Framework for Container-based Clouds

Karthick Rajamani, Alexandre Ferreira, Juan Rubio
Optimized Cloud Infrastructure, IBM Research

Wes Felter
IBM Cloud Innovation Lab

{karthick,apferrei,rubioj,wmf}@us.ibm.com
Overview

• What is Spyre?
• Resource management with Spyre
• Performance evaluation
• Status and next steps
• Extending Tenant SLA models – discussion
Containers offer better performance than VMs

Sysbench with MySQL

Source: An Updated Performance Comparison of Virtual Machines and Linux Containers – Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio
What is Spyre?

Optimized foundation for the container-based cloud

- Containers are fundamental unit of computation (not container in VM)
- Superior resource isolation and performance (tail latency) for tenant/performance-sensitive services – resource-isolated slices.
- Support resource-sharing among containers used as side-cars (running within same slice).
- Avoid multi-tenant dockerd issue – each client (slice) can have their own dockerd.
- Can be used with any container eco-system – we have experimented to date with Docker.
Spyre Goals

• Predictable performance (including tail)
  – Strong isolation (e.g., dedicated physical cores) with *slices*
  – Allocate resources using *real* units (say Ghz not abstract compute units)
  – Unique use/configuration of cgroups

• Vertical scaling
  – Grow containers while running (e.g., add cores/RAM)
  – Subletting: spot market (like a CloudBnB)

• High performance
  – Base unit is containers
  – Optimize storage & network I/O

  *e.g., eliminate NAT and replace AUFS with block storage*
Resource management with Spyre

• Key concept: Slices
  • Dedicated resources for predictable/guaranteed performance
    • Dedicated physical cores
    • Dedicated RAM
Challenges to sharing core resources

- Shared cores result in variable impact on performance
  - Significant, difficult to predict impact for tenant workload
  - Difficult to predict returns for provider

- Data taken on POWER8 processor which has dedicated L1, L2, L3 cache per core
Resource management with Spyre

- Key concept: Slices
  - Dedicated resources for predictable performance
    - Dedicated physical cores
    - Dedicated RAM
  - Guaranteed minimum network bandwidth
  - Multiple vNICs, IP addresses, block storage (optional)
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>cores</th>
<th>RAM (GB)</th>
<th>L3 (MB)</th>
<th>Net. BW</th>
<th>Price/hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDW-2GB-HT</td>
<td>Broadwell 2GHz</td>
<td>1/4</td>
<td>2</td>
<td>-</td>
<td>0.31</td>
<td>$0.04</td>
</tr>
<tr>
<td>BDW-4GB-1T</td>
<td>Broadwell 2GHz</td>
<td>1/2</td>
<td>4</td>
<td>-</td>
<td>0.63</td>
<td>$0.06</td>
</tr>
<tr>
<td>BDW-8GB-1C</td>
<td>Broadwell 2GHz</td>
<td>1</td>
<td>8</td>
<td>1.5</td>
<td>1.25</td>
<td>$0.10</td>
</tr>
<tr>
<td>BDW-16GB-2C</td>
<td>Broadwell 2GHz</td>
<td>2</td>
<td>16</td>
<td>3.0</td>
<td>2.50</td>
<td>$0.22</td>
</tr>
<tr>
<td>BDW-24GB-3C</td>
<td>Broadwell 2GHz</td>
<td>3</td>
<td>24</td>
<td>4.5</td>
<td>3.75</td>
<td>$0.33</td>
</tr>
<tr>
<td>BDW-32GB-4C</td>
<td>Broadwell 2GHz</td>
<td>4</td>
<td>32</td>
<td>6.0</td>
<td>5.00</td>
<td>$0.44</td>
</tr>
<tr>
<td>BDW-40GB-5C</td>
<td>Broadwell 2GHz</td>
<td>5</td>
<td>40</td>
<td>7.5</td>
<td>6.25</td>
<td>$0.55</td>
</tr>
<tr>
<td>BDW-48GB-6C</td>
<td>Broadwell 2GHz</td>
<td>6</td>
<td>48</td>
<td>9.0</td>
<td>7.50</td>
<td>$0.66</td>
</tr>
<tr>
<td>BDW-56GB-7C</td>
<td>Broadwell 2GHz</td>
<td>7</td>
<td>56</td>
<td>10.5</td>
<td>8.75</td>
<td>$0.77</td>
</tr>
<tr>
<td>BDW-64GB-8C</td>
<td>Broadwell 2GHz</td>
<td>8</td>
<td>60</td>
<td>12.0</td>
<td>10.00</td>
<td>$0.80</td>
</tr>
<tr>
<td>P8-4GB-2T</td>
<td>Power8 3.x GHz</td>
<td>1/4</td>
<td>4</td>
<td>-</td>
<td>0.25</td>
<td>$0.06</td>
</tr>
<tr>
<td>P8-8GB-4T</td>
<td>Power8 3.x GHz</td>
<td>1/2</td>
<td>8</td>
<td>-</td>
<td>0.5</td>
<td>$0.11</td>
</tr>
<tr>
<td>P8-16GB-1C</td>
<td>Power8 3.x GHz</td>
<td>1</td>
<td>16</td>
<td>8.0</td>
<td>1</td>
<td>$0.20</td>
</tr>
</tbody>
</table>
Observations from other work

  – Latency critical workloads need dedicated/isolated resources, distinct from those allowed to be assigned for batch workloads

• Microservices require stronger focus around component-service tail latencies
  – Increased probability of impact on composite service latency.

Spyre-slice frame-work of value also to latency-sensitive cloud services.
Resource management with Spyre

- **Key concept: Slices**
  - Dedicated resources for predictable performance
    - Dedicated physical cores
    - Dedicated RAM
  - Guaranteed minimum network bandwidth
  - Multiple vNICs, IP addresses, block storage (optional)

- **Implemented using cgroups & systemd units**
  - Note: systemd does not yet support dedicated cores (cpusets), custom script implements it.

- **Multiple containers per slice (similar to Kubernetes pod/Carina segment)**
  - Allows intra-customer sharing of resources
Resource View

Slice

- Cores
- Memory
- iSCSI volumes

Host

- vNICs
- Memory
- NICs (time-shared)
- Possibly no local storage
Slice — tenant view

Host A

- sshd
- dockerd

Docker container X

Docker container Y

Docker container Z

eth0
Host — software view
Slice benefit analysis with an in-memory database workload

- DB2 BLU (in-memory database)
  - AGG_COL
    - includes up to 10 concurrent streams of SQL queries
    - used to emulate background, interfering job
  - REPORT_COL
    - includes up to 10 concurrent streams of SQL queries
    - used to emulate foreground job.
- 4 instances of 100GB datasets with 3 REPORT_COL and 1 AGG_COL executed concurrently in
  - 4 Docker containers on Host
  - 4 Docker containers, each within own slice (6-core) on Host
- 24-core POWER8-S824 machine (2 6-core dies per socket, 2 sockets) with 512GB of memory spread evenly among the dies.
- All databases are resident on iSCSI volumes
- 2 runs done for both container-only and containers-within-slice scenarios
  - 6 data points for REPORT_COL, 2 data points for AGG_COL for each scenario
REPORT_COL average

Average Stream Execution Time (REPORT_COL)

Average better or same with slices.
REPORT_COL worst-case performance

Slices improve worst-case performance i.e. lower tail latency (lower ratio of max to average)
AGG_COL average

Average Stream Execution Time (AGG_COL)

AGG_COL benefits from stealing resources, i.e., sees lower performance when constrained within slice.
AGG_COL worst-case performance

Ratio of Max. exec. time across runs by average exec. time (AGG_COL)

Lower variation of runtimes with slices.
caveat: only two data points behind each bar.
Spyre Status

sliced in Linux on x86 and POWER:

• Interface
  – Simple REST API supporting slice create, query, resize, delete, and to query system for resources available/free
  – Returns and accepts JSON

• Capability
  – Provides CPU (core, cache) isolation
  – Automatic memory affinity with CPU
  – Vertical scaling

• Implementation
  – Python
  – Systemd, cgroups, cpuset
  – Pflask for outer container
  – Slice has own IP, ssh access with public key

Opening project to community
Next Steps

• *Spyred* implementation for stand-alone cluster.
• Memory bandwidth control (IBM POWER8) and shared-cache control (Intel Haswell+) – hardware-specific.
• Networking design and network bw control work.
• Storage design.
• Integration with broader eco-system: Machine +Swarm, Kubernetes, Mesos... (?)
• *Extending tenant SLA models*
Extending Tenant-Slice service models

Guaranteed Resources

Performance Isolation

Dedicated
High priority
(CPU sets)
Extending Tenant-Slice service models – Vertical Resizing

Guaranteed Resources

- Pays for Current and a premium to go up to Max.

Dedicated
High priority
(CPU sets)

Max
Current

Dedicated
Low priority
(CPU sets)

Min
Current

Pays for Current and a discount for allowing to be taken down to Min.
Extending Tenant-Slice service models – Increasing density

- Guaranteed Resources
  - Shared High-Priority (CPU shares)
  - Dedicated High-priority (CPU sets)
  - Dedicated Low-Priority (CPU sets)

- Performance Isolation

  Gets a discount for tolerating jitter, potentially higher tail latencies.
Extending Tenant-Slice service models – Increasing density

- Guaranteed = Requested resource averaged over some time interval
- Dedicated High priority (CPU sets)
- Dedicated Low-Priority (CPU sets)
- Shared High-Priority (CPU shares)
- Shared Low-Priority (CPU shares)

Guaranteed = Requested resource when occasionally active (enables provider to overcommit); gets discount for not needing requested resources all the time.
Discussion

• How important is dynamic resizing ability – both growing up and ability to pay for a lower minimum?

• Is dynamic resizing applicable to memory
  – Can applications deal with some of their allocated memory being moved to swap?
  – Will high-speed swap (SSD/NVME backed) help?

• If a system supports both dedicated and shared is there need for high/low priority sub-classes?

• Any user classes not covered by these models?

• Any other comments, questions?
Thank you

IBM Research is hiring in Cloud Infrastructure and Data centers area.

If interested please contact me,
Email: karthick@us.ibm.com