Towards a Carrier Grade ONAP Platform Performance and Isolation Architectural Evolution

Key Contributors: Bin Yang, Gil Hellmann

Agenda

- What is Performance Awareness and Why
- Scenarios
- Categories
  - Hardware features
  - Platform Resources and Services

What is Performance Awareness and Why

- Whenever Performance & Isolation Matters
  - Increased or dedicated allocation of the resources
  - Lower utilization
- Whenever Utilization Matters
  - Consolidate workloads to share resources as much as possible
  - lower performance
- Can we have both simultaneous?
  - Make the best use of infrastructure's enhanced features.

- *Can we make this simple and portable between different infrastructure providers?*
  - Multi-Cloud project could help by modeling/discovering/managing the infrastructure's capabilities and resources
  - Make ONAP be aware of infrastructure's enhanced features through common data models
    - Enhanced features includes advanced capabilities, resources and services that the infrastructure platforms expose to ONAP

Scenario 1: NUMA awareness

- Think about NUMA awareness
  - Improve performance by reducing memory access latency
  - OpenStack Flavor extra-specs is the current approach to make use of it
  - VNF vendors design and implement VNFs based on assumptions of NUMA topologies
    - But how to cope with various NUMA topologies which VNF vendors don't know yet?
    - VNF vendors could make wrong assumption because they have limited information about NUMA topologies of target infrastructures.
    - Orchestrators cannot help either since they lack insight into VNF design/implementation.
    - **To make best usage of NUMA awareness, VNF vendor/Orchestrator/Infrastructure providers need to collaborate with more precise modeling**

Scenario 1: NUMA awareness (Cont.)

- With this case, the model should enable:
  - VNF vendor specifies which specific vCPUs are more sensitive to memory access latency, and how much of memory are required
  - Infrastructure providers discover and expose the available NUMA topologies to orchestrator
  - The orchestrator matches the requirement of VNF and the available NUMA topologies, comes up with proper NUMA spec, then instantiate the VNFs with these NUMA spec.

NUMA topologies – Exemplary model

Exemplary model for Infrastructure NUMA topologies

```
NUMA_topologies:
  NUMA_topology1:
      pservers:
        edge to pserver1,
        edge to pserver2,
      numa_nodes:
       node0:
        id: 0
          vcpus: 24
        mem_size: 65535 MB
       node1:
        id: 1
        vcpus: 24
        mem_size: 65535 MB
```

```
pserver:
  NUMA_used:
      numa_nodes:
       node0:
        id: 0
          vcpus: 6
        mem_size: 35000 MB
       node1:
        id: 1
        vcpus: 12
        mem_size: 49152 MB
```

NUMA requirement – Exemplary TOSCA Models

VM NUMA requirement

```
VMD1:
 capabilities:
  compute:
   properties:
    mem_size: 4096 MB
    num_cpus: 4
    numa_nodes:
     node0:
      id: 0
        vcpus: [0,1]
      mem_size: 1024 MB
        latency_sentitive: false
     node1:
      id: 1
      vcpus: [2,3]
      mem_size: 3072 MB
        latency_sentitive: true
```

Categories

- Memory Access Latency and Throughput
  - NUMA affinity
  - PCI NUMA affinity
  - Huge Page
  - RDT
- Computation Intensive Workloads Optimization
  - CPU pinning
  - CPU thread policy
  - Shared VCPU ID
- Networking throughput and Latency
  - SRIOV
  - PCI pass-through for NIC
  - DPDK based vSwitch
  - vSwitch NUMA affinity

Categories (Cont.)

- Instruction Sets
  - vCPU model
- Accelerators
  - PCI pass-through for encryption/compression
  - PCI pass-through for transcoding
- Security and Isolation
  - vTPM
  - TXT
- Platform Services/Resources
  - Live migration
  - Health monitoring/auto-healing
  - Firewall
  - DNS service

Q&A

Thank you

Backup

A way of Modeling Compute Profiles beyond Individual Properties

```
compute_profile_xyz:
  description: >-
    some compute profile.
  compute_dependencies:
    dpdk:
      string: { get_input: dpdk.version }
    mem_page_size:
      string: large
      optional: true
    sr-iov:
      boolean: true
    cpu_allocation:
      string_map:
        cpu_affinity: dedicated
```

Compute Profiles can be defined by the operator using TOSCA to create a "bucket" of compute dependencies. The profiles are suppose to simplify the definitions of Compute Dependencies, as in a lot of deployments there are many VDUs with the same required EPA features running on the same infrastructure hardware. If a Compute Profile is not used, then all Compute Dependencies have to specified for a given VDU.

TOSCA modeling example for VDU with Compute profile

```
xyz_vdu:
  description: >-
    The “xyz" VDU provides feature xyz.
  type: tosca.nodes.nfv.VDU
  interfaces:
    Standard:
      configure: scripts/vdu/xyz_configure.sh # included in the CSAR
  requirements:
    - dependency:
        node: xyz_host # our Compute node
        relationship:
          type: vnfsdk.DependsOn
          properties:
            # For the “xyz" VDU we are choosing a profile that already includes
            # various compute dependencies:
            compute_profile: compute_profile_xyz
```