KubeCon | CloudNativeCon

Europe 2022

WELCOME TO VALENCIA

KubeCon CloudNativeCon
Europe 2022

Network-aware Scheduling in Kubernetes

José Santos (Postdoctoral Researcher, Ghent University - imec)
Chen Wang (Research Scientist, IBM)

Agenda

1. Background & Motivation
2. Network-aware framework - Design overview
3. Implementation details
4. Network-Aware Scheduling Plugins
5. Demo – Deploying Online Boutique
6. Challenges & Lessons Learned
7. Acknowledgements

Background & Motivation

- The K8s scheduler typically focuses on Resource Efficiency (e.g., CPU and Memory).
- No contextual awareness about application dependencies or infrastructure topology.
- Low latency plays a major role for several applications (e.g., IoT, video streaming).

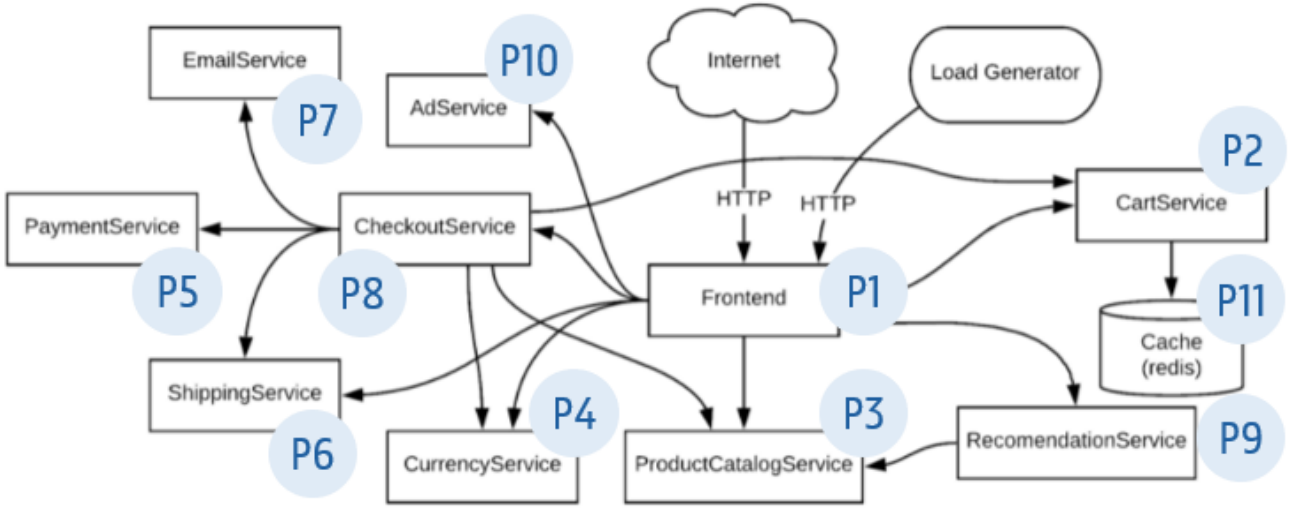
How can we do better?

Consider latency and network bandwidth in the scheduling process. But how?

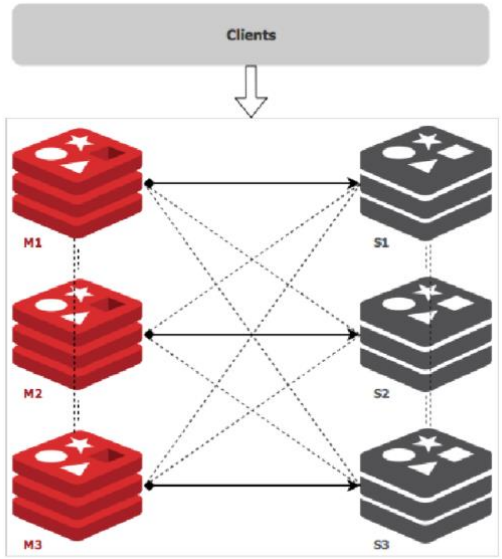
Proposed solution

- Develop a framework with additional scheduling plugins that consider:
  - Application characteristics: microservice dependencies play a major role.
  - Infrastructure Topology: weights among cluster nodes based on different metrics.

Online Boutique



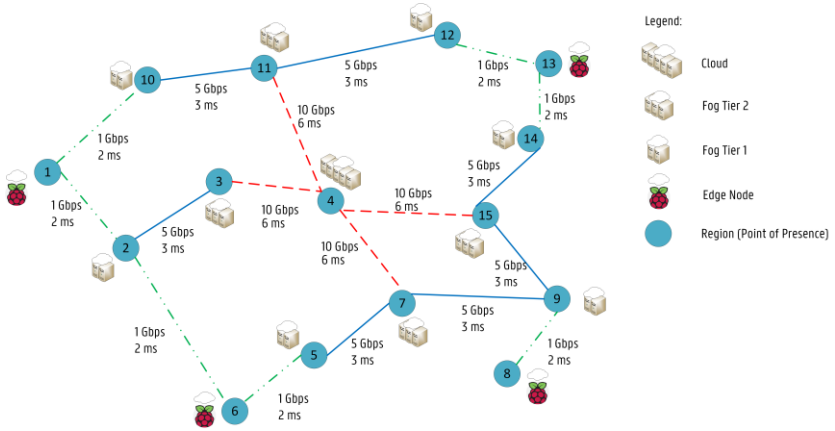
Redis cluster



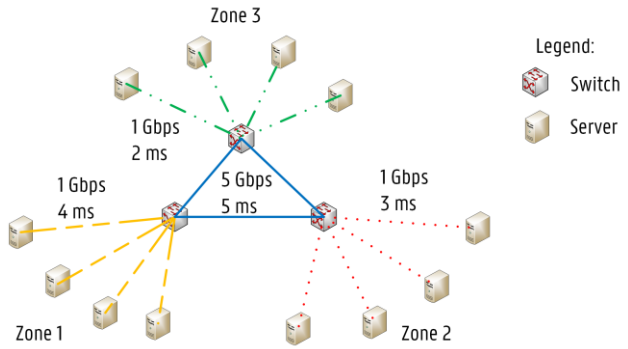
Use Cases / Topologies

- All kinds of topologies would benefit from this framework.
- High latency is a big concern in multi-region clusters.
- Even a small-scale cluster would benefit from latency-aware scheduling decisions.
- Latency in Data centers is a critical requirement for certain applications (e.g., financial).

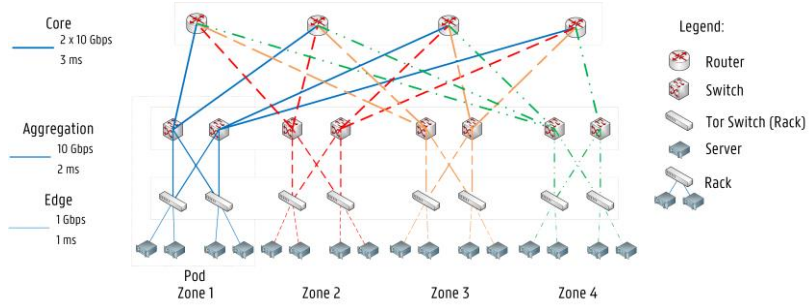
Multi-region



Multi-Zone Cluster



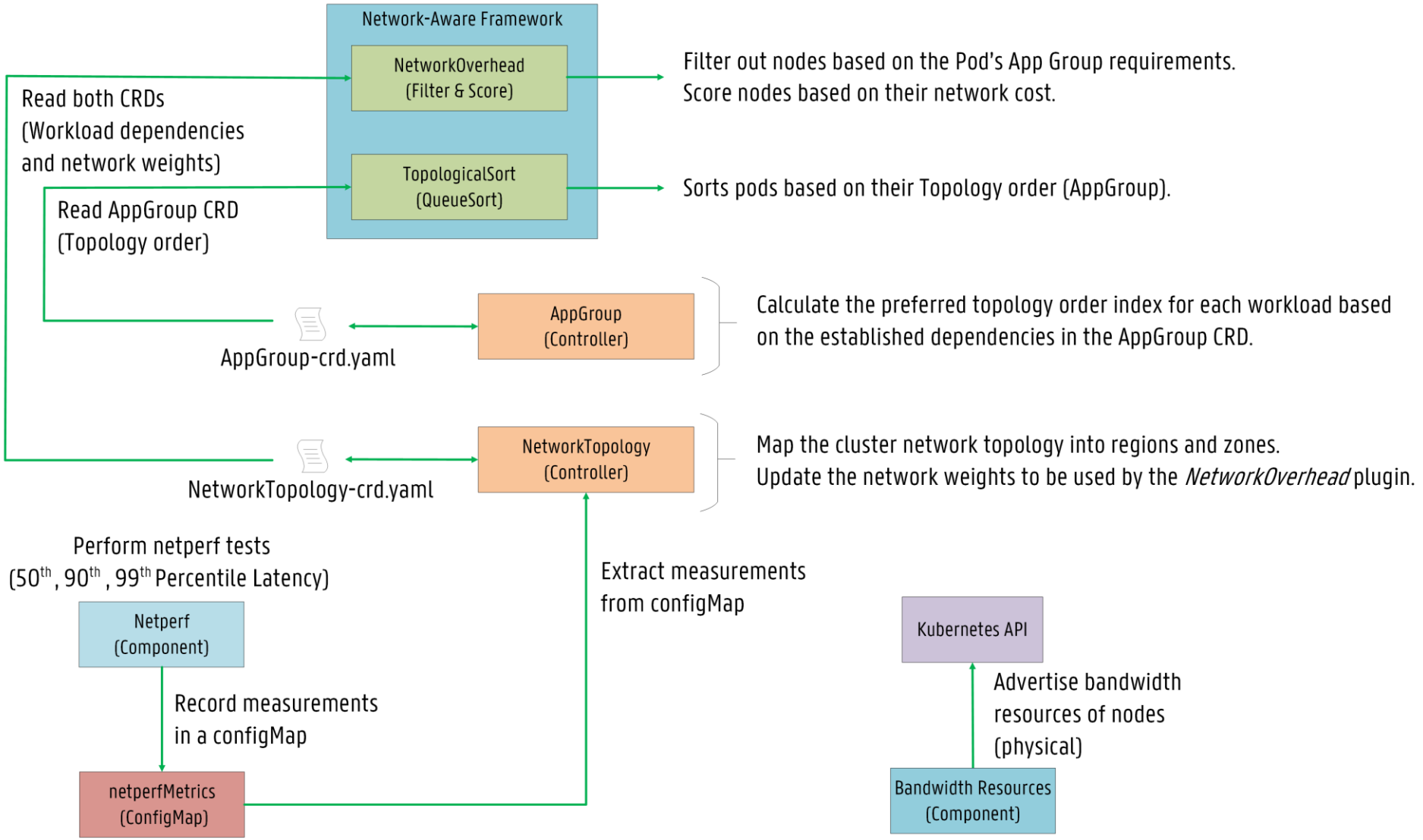
Data Center



Design & Implementation

Main ideas

Network-aware framework – Design Overview



This design introduces an end-to-end solution to model/weight a cluster's network latency and topological information, and leverage that to optimize the scheduling of latency- and bandwidth-sensitive workloads.

Netperf Component: https://github.com/jpedro1992/pushing-netperf-metrics-to-prometheus
Development repo: https://github.com/jpedro1992/scheduler-plugins/tree/KepDevWithNTController

The inclusion of bandwidth in the scheduling process

Advertise bandwidth resources on all nodes (physical)

```
PATCH /api/v1/nodes/<your-node-name>/status HTTP/1.1
Accept: application/json
Content-Type: application/json-patch+json
Host: k8s-master:8080

[
  {
    "op": "add",
    "path": "/status/capacity/network.aware.com~1bandwidth",
    "value": "10000000000"
  }
]
```

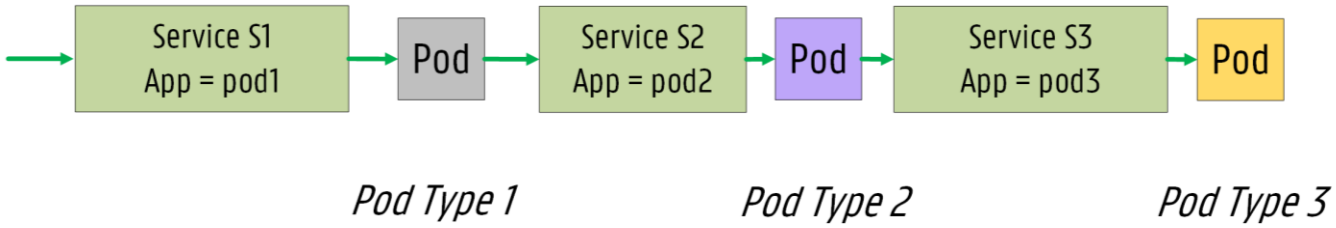
The bandwidth CNI plugin supports pod ingress and egress traffic shaping

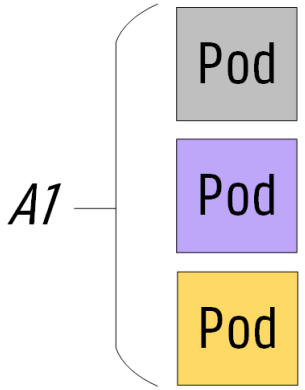
```
# Example Pod deployment with bandwidth limitations
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/ingress-bandwidth: 1M
    kubernetes.io/egress-bandwidth: 1M
```

```
# Example Pod deployment:
# bandwidth requests (Extended Resources)
# bandwidth limitations (bandwidth CNI plugin)
apiVersion: v1
kind: Pod
metadata:
  name: network-aware-bandwidth-example
  annotations:
      kubernetes.io/ingress-bandwidth: 10M
      kubernetes.io/egress-bandwidth: 10M
spec:
  containers:
  - name: network-aware-bandwidth-example
    image: example
    resources:
      requests:
        network.aware.com/bandwidth: 10000000 # 10M
      limits:
        network.aware.com/bandwidth: 10000000 # 10M
```

https://kubernetes.io/docs/tasks/administer-cluster/extended-resource-node/
https://kubernetes.io/docs/tasks/configure-pod-container/extended-resource

AppGroup CRD - Example





Spec

```yaml
# Example App Group CRD spec
apiVersion: scheduling.sigs.k8s.io/v1alpha1
kind: AppGroup
metadata:
  name: a1
spec:
  numMembers: 3
  topologySortingAlgorithm: KahnSort
  workloads:
    - workload:
        kind: Deployment
        apiVersion: apps/v1
        namespace: default
        name: P1
      dependencies:
        - workload:
            kind: Deployment
            apiVersion: apps/v1
            namespace: default
            name: P2
          minBandwidth: "100Mi"
          maxNetworkCost: 30
    - workload:
        kind: Deployment
        apiVersion: apps/v1
        namespace: default
        name: P2
      dependencies:
        - workload:
            kind: Deployment
            apiVersion: apps/v1
            namespace: default
            name: P3
          minBandwidth: "250Mi"
          maxNetworkCost: 20
    - workload:
        kind: Deployment
        apiVersion: apps/v1
        namespace: default
        name: P3
```

Bandwidth and network requirements

Workload (Pod) dependencies with specific affinities

Status

Records preferred Topology order

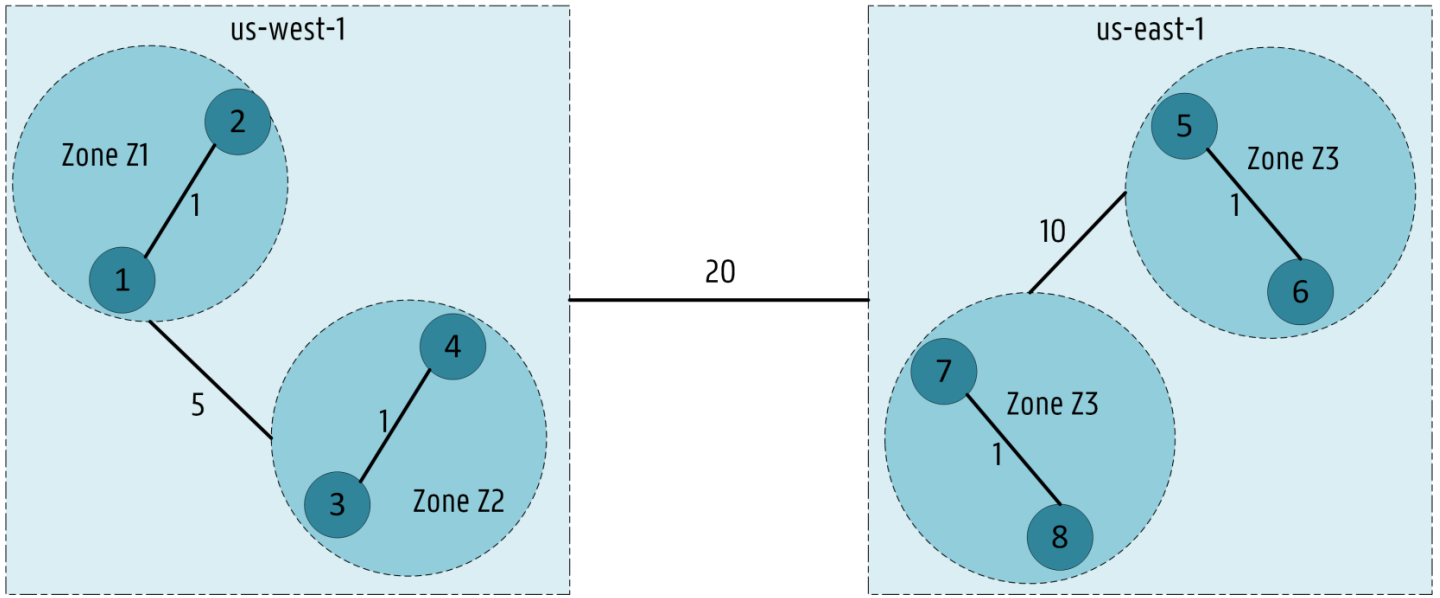
NetworkTopology CRD - Example

Spec

```yaml
# Example Network CRD
apiVersion: scheduling.sigs.k8s.io/v1alpha1
kind: NetworkTopology
metadata:
  name: net-topology-test
  namespace: default
spec:
  configMapName: "netperfMetrics"
  weights:
    # Region label: "topology.kubernetes.io/region"
    # Zone Label:   "topology.kubernetes.io/zone"
    # 2 Regions:  us-west-1
    #             us-east-1
    # 4 Zones:    us-west-1: z1, z2
    #             us-east-1: z3, z4
    - name: "UserDefined"
      costList: # Define weights between regions or between zones
        - topologyKey: "topology.kubernetes.io/region" # region costs
          originCosts:
            - origin: "us-west-1"
              costs:
                - destination: "us-east-1"
                  bandwidthCapacity: "10Gi"
                  networkCost: 20
            - origin: "us-east-1"
              costs:
                - destination: "us-west-1"
                  bandwidthCapacity: "10Gi"
                  networkCost: 20
        - topologyKey: "topology.kubernetes.io/zone" # zone costs
          originCosts:
            - origin: "z1"
              costs:
                - destination: "z2"
                  bandwidthCapacity: "1Gi"
                  networkCost: 5
            - origin: "z2"
              costs:
                - destination: "z1"
                  bandwidthCapacity: "1Gi"
                  networkCost: 5
            - origin: "z3"
              costs:
                - destination: "z4"
                  bandwidthCapacity: "1Gi"
                  networkCost: 10
            - origin: "z4"
              costs:
                - destination: "z3"
                  bandwidthCapacity: "1Gi"
                  networkCost: 10
```

Region costs

Zone costs

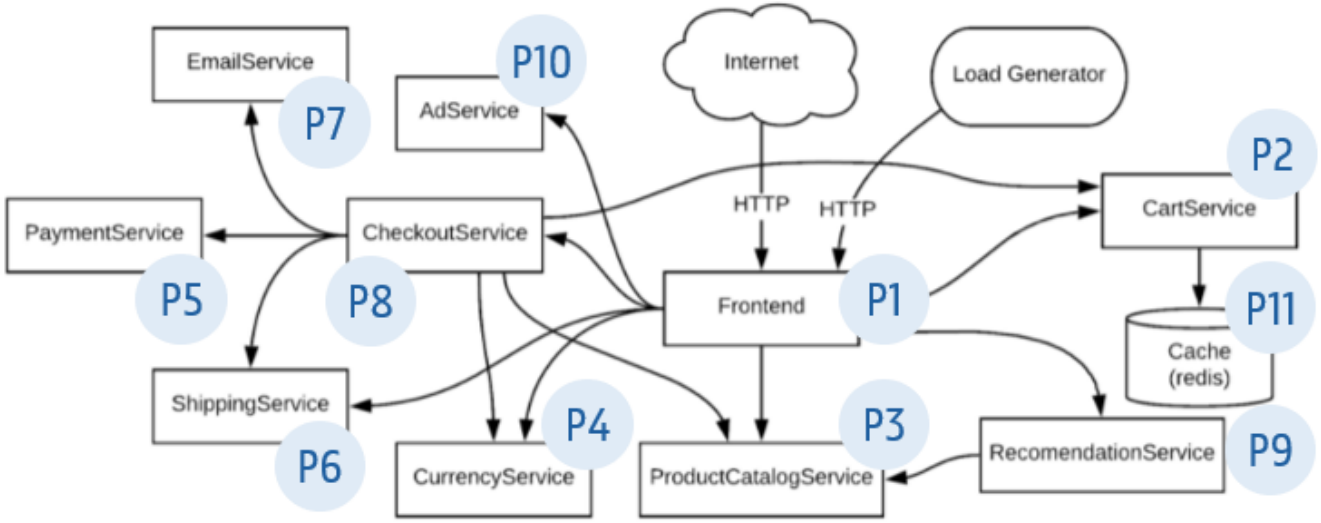


Network-Aware Scheduling Plugins

Examples

TopologicalSort (QueueSort) - the impact of queue sorting pods

If pods have several microservice dependencies, how to select the first pod to be allocated?
Consider the OnlineBoutique application (P1 – P11) shown below. Which pod should be deployed first?



Potential Solution

Develop a Queue Sort plugin that sorts pods based on Topological Sorting.

Pods belonging to an AppGroup should be sorted based on their topology information.

Significant differences are obtained depending on which topology algorithm is selected, as shown in the next slide.

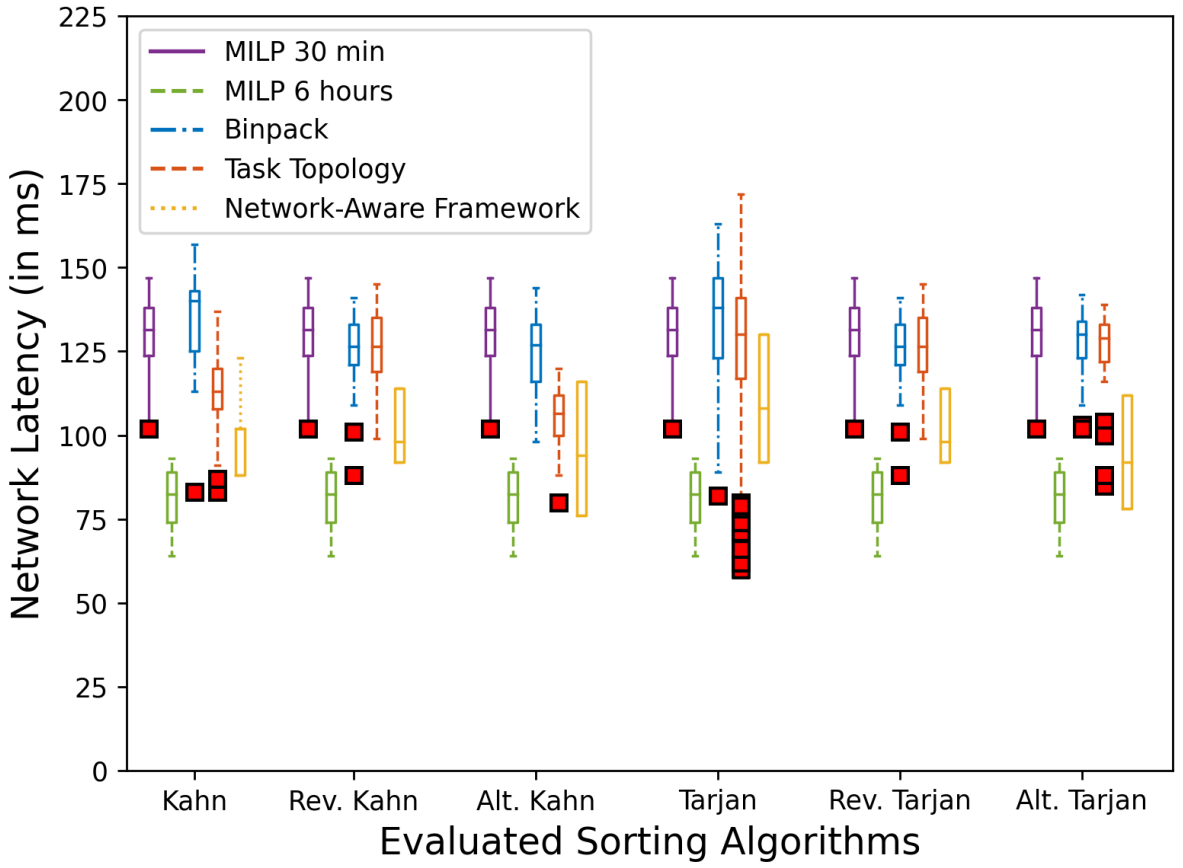
TopologicalSort (QueueSort) - Example

1. Compare Pods based on their AppGroup.
2. If equal, get AppGroup info.
3. Get pod index of both pods.
4. Lower order is better. Should be allocated first.
5. If Pods do not belong to the same AppGroup, follow the strategy of the QoS plugin.

Supported algorithms

Sorting Algorithm	Topological order
Kahn	P1, P10, P9, P8, P7, P6, P5, P4, P3, P2, P11.
Alternate Kahn	P1, P11, P10, P2, P9, P3, P8, P4, P7, P5, P6.
Reverve Kahn	P11, P2, P3, P4, P5, P6, P7, P8, P9, P10, P1.
Tarjan	P1, P8, P7, P5, P4, P2, P11, P9, P10, P6, P3.
Alternate Tarjan	P1, P3, P8, P6, P7, P10, P5, P9, P4, P11, P2.
Reverve Tarjan	P3, P6, P10, P9, P11, P2, P4, P5, P7, P8, P1.

We achieved lower latency with Kahn, Alt. Kahn and Alt. Tarjan for the Online Boutique application.



NetworkOverhead (Filter & Score) – towards low latency allocation schemes

Workload dependencies established in the AppGroup CR must be respected.
Several dependencies may exist since multiple pods can be already deployed on the network.

Extension Point: Filter → nodes are filtered out if they cannot support the network requirements of the Pod's AppGroup.

Extension Point: Score → scored based on network weights ensuring network latency is minimized for pods belonging to the same AppGroup.

NetworkOverhead – Example (AppGroup)


```yaml
# Example App Group CRD spec
apiVersion: scheduling.sigs.k8s.io/v1alpha1
kind: AppGroup
metadata:
  name: a1
spec:
  numMembers: 3
  topologySortingAlgorithm: KahnSort
  workloads:
    - workload:
        kind: Deployment
        name: P1-deployment
        selector: P1
        apiVersion: apps/v1
        namespace: default
      dependencies:
        - workload:
            kind: Deployment
            name: P2-deployment
            selector: P2
            apiVersion: apps/v1
            namespace: default
          minBandwidth: "100Mi"
          maxNetworkCost: 30
    - workload:
        kind: Deployment
        name: P2-deployment
        selector: P2
        apiVersion: apps/v1
        namespace: default
      dependencies:
        - workload:
            kind: Deployment
            name: P3-deployment
            selector: P3
            apiVersion: apps/v1
            namespace: default
          minBandwidth: "250Mi"
          maxNetworkCost: 20
    - workload:
        kind: Deployment
        name: P3-deployment
        selector: P3
        apiVersion: apps/v1
        namespace: default
```

AppGroup CRD

A1: P1, P2, P3

Dependencies

P1: P2

P2: P3

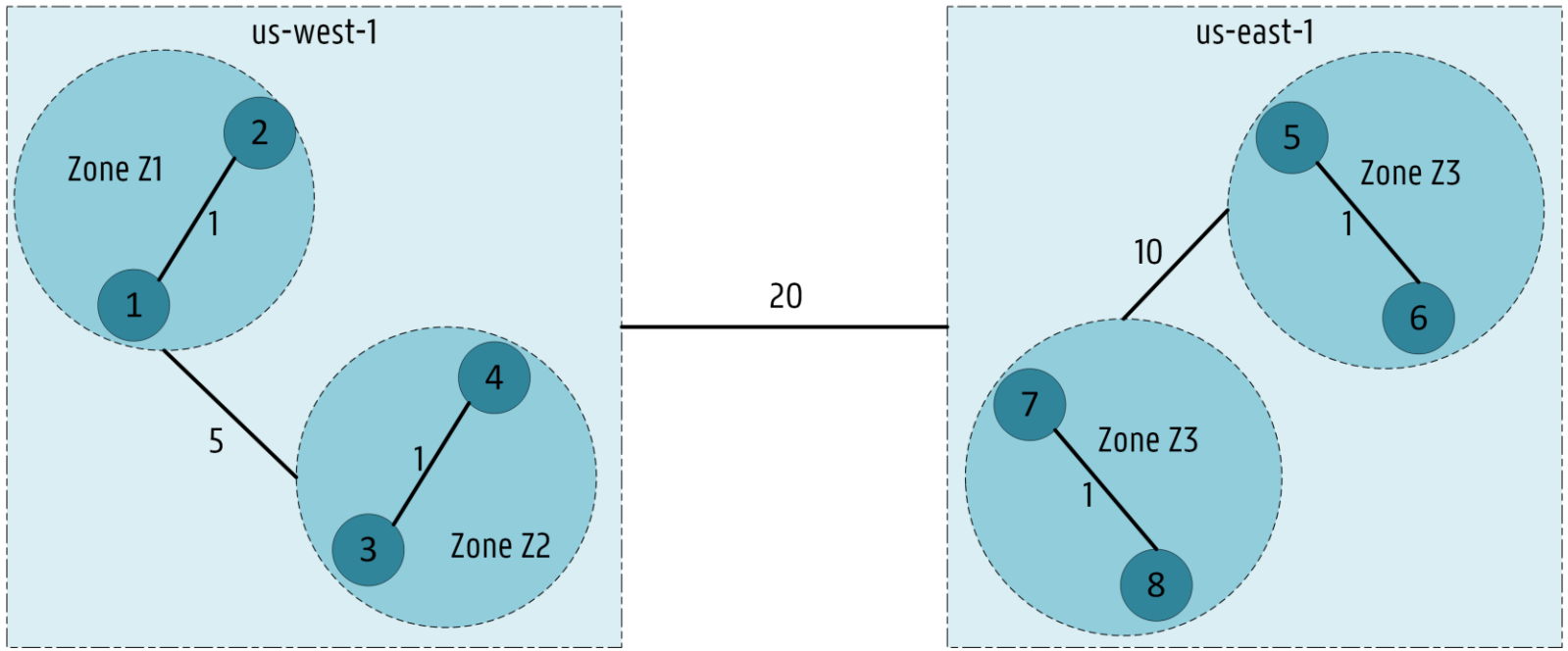
NetworkOverhead – Example (NetworkTopology)


```yaml
# Example Network CRD
apiVersion: scheduling.sigs.k8s.io/v1alpha1
kind: NetworkTopology
metadata:
  name: net-topology-test
  namespace: default
spec:
  configmapName: "netperfMetrics"
  weights:
    # Region label: "topology.kubernetes.io/region"
    # Zone Label:   "topology.kubernetes.io/zone"
    # 2 Regions:  us-west-1
    #             us-east-1
    # 4 Zones:    us-west-1: z1, z2
    #             us-east-1: z3, z4
    - name: "UserDefined"
      topologyList: # Define weights between regions or between zones
        - topologyKey: "topology.kubernetes.io/region" # region costs
          originList:
            - origin: "us-west-1"
              costList:
                - destination: "us-east-1"
                  bandwidthCapacity: "10Gi"
                  networkCost: 20
            - origin: "us-east-1"
              costList:
                - destination: "us-west-1"
                  bandwidthCapacity: "10Gi"
                  networkCost: 20
        - topologyKey: "topology.kubernetes.io/zone" # zone costs
          originList:
            - origin: "z1"
              costList:
                - destination: "z2"
                  bandwidthCapacity: "1Gi"
                  networkCost: 5
            - origin: "z2"
              costList:
                - destination: "z1"
                  bandwidthCapacity: "1Gi"
                  networkCost: 5
            - origin: "z3"
              costList:
                - destination: "z4"
                  bandwidthCapacity: "1Gi"
                  networkCost: 10
            - origin: "z4"
              costList:
                - destination: "z3"
                  bandwidthCapacity: "1Gi"
                  networkCost: 10
```

NetworkTopology CRD
net-topology-test

Topology Info:

2x Regions: us-west-1, us-east-1
4x Zones: Z1 – Z4
8x Nodes: N1 – N8



NetworkOverhead – Example (Filter)

A1: P1, P2, P3
Candidate Nodes: N1 – N8

Workload to schedule: P1

P2 is a dependency!

Workload allocations:

P2 deployed on N1!

Nodes that unmet a higher number of dependencies are filtered out to reduce the number of nodes being scored.

For this example, nodes that do not respect the network cost requirement between P1 and P2 (i.e., maxNetworkCost: 15) are filtered out.

Pod to schedule

Pod
P1

Application Group

Workload	Kind
P1	Deployment
P2	Deployment
P3	Deployment

Pod dependencies

Workload	Pod	maxNetworkCost	Hostname
P2	P2-one	15	N1

Filter Operation

Nodes	Region	Zone	Network Cost to N1	Result
N1	us-west-1	Z1	0	PASS
N2	us-west-1	Z1	1	PASS
N3	us-west-1	Z2	5	PASS
N4	us-west-1	Z2	5	PASS
N5	us-east-1	Z3	20	FILTER
N6	us-east-1	Z3	20	FILTER
N7	us-east-1	Z4	20	FILTER
N8	us-east-1	Z4	20	FILTER

NetworkOverhead – Example (Score)

A1: P1, P2, P3
Candidate Nodes: N1 – N4

Calculate the accumulated shortest path cost for all candidate nodes.

Normalize (between 0 and 100) the accumulated cost for all nodes.

Nodes with lower costs are favored since lower costs correspond to lower latency

Pod to schedule

Pod
P1

Network Costs

SameHostname	SameZone
0	1

Application Group

Workload	Kind
P1	Deployment
P2	Deployment
P3	Deployment

Maximum Score

MAX Score
100

Pod dependencies

Workload	Hostname
P2	N1

Allocations

Workload	Pod	Hostname
P2	P2-one	N1
P2	P2-sec	N3

Scoring Operation

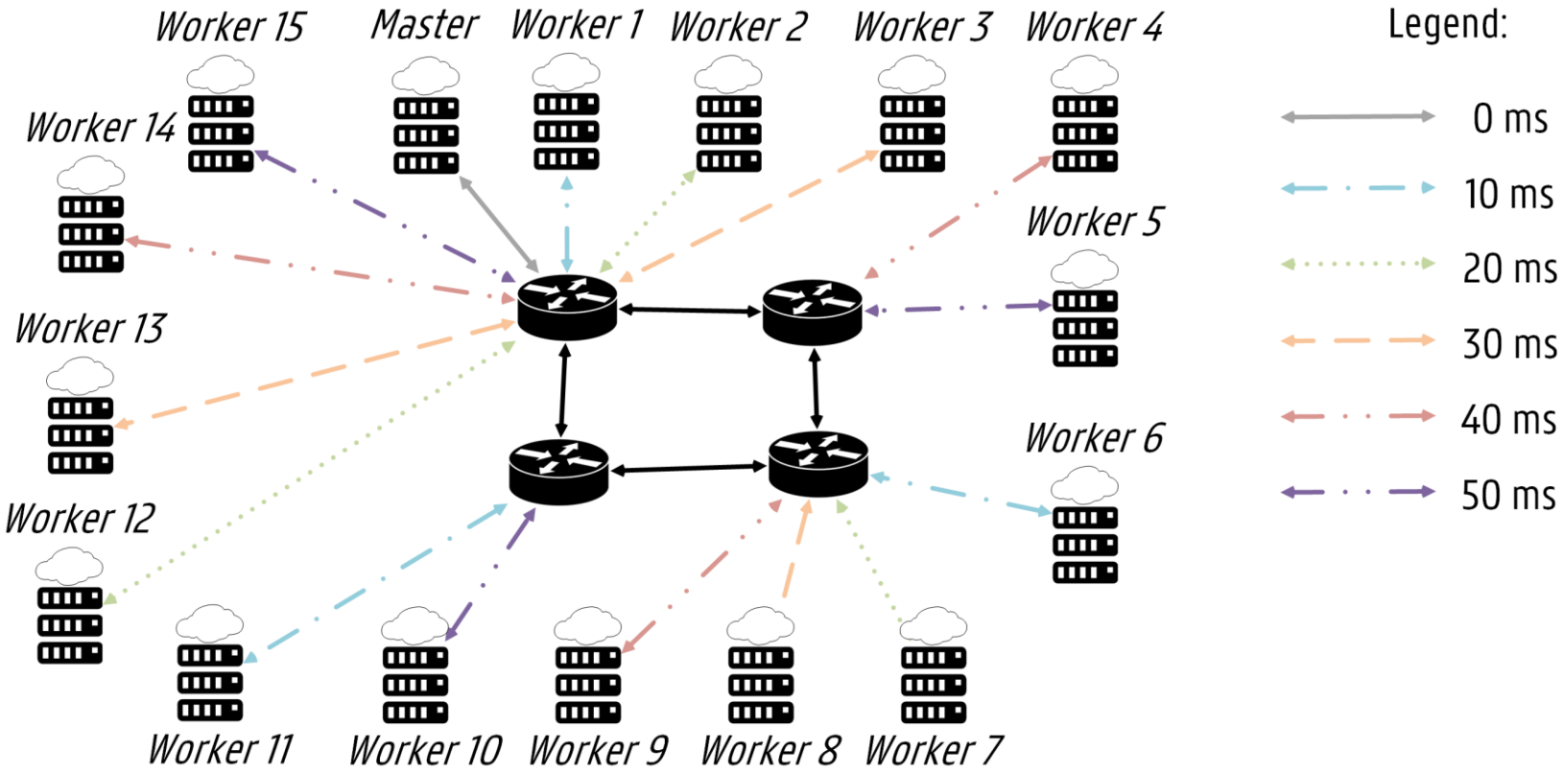
Nodes	Region	Zone	Accumulated cost	MAX Cost	MIN Cost	Normalized Cost	Node Score
N1	us-west-1	z1	0	5	0	0.00	100
N2	us-west-1	z1	1	5	0	20.00	80
N3	us-west-1	z2	5	5	0	100.00	0
N4	us-west-1	z2	5	5	0	100.00	0

Demo

Deploying Online Boutique

Demo Setup

- A K8s cluster with 16 nodes (1 master and 15 workers).
- All nodes belong to the same region (belgium), but each node is in a different zone (i.e., master, z1, .., z15).
- Varying delays are emulated on network connections via Traffic Control (TC).



Challenges & Lessons Learned

- The developed plugins significantly reduce the expected network latency in K8s clusters.
- Several pod "grouping" definitions exist: PodGroup, AppGroup, …
- The developed plugins do not add significant overhead (Exec. Time: below 1 sec for 10000 nodes).

Looking for contributors and engagement from the community! ☺

Kubernetes sig-scheduling repo: https://github.com/kubernetes-sigs/scheduler-plugins
Network-aware framework KEP: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/kep/260-network-aware-scheduling
Initial PR: https://github.com/kubernetes-sigs/scheduler-plugins/pull/348

Acknowledgements

KubeCon CloudNativeCon Europe 2022

Chen Wang
IBM Research

Asser Tantawi
IBM Research

Tim Wauters
Ghent University- imec

Prof. Filip De Turck
Ghent University - imec

Kubernetes Sig-Scheduling

Q&A

José Santos

Postdoctoral Researcher (PhD)
IDLab - Ghent University - imec

josepedro.pereiradossantos@UGent.be

https://github.com/jpedro1992/

https://www.linkedin.com/in/jpedro1992/

Chen Wang

Research Scientist
IBM

chen.wang1@ibm.com

wangchen615@github

https://www.linkedin.com/in/chenw615/

Kubernetes sig-scheduling repo: https://github.com/kubernetes-sigs/scheduler-plugins
Network-aware framework KEP: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/kep/260-network-aware-scheduling
Initial PR: https://github.com/kubernetes-sigs/scheduler-plugins/pull/348

Additional Slides

Q&A

AppGroup CRD

Spec

```
spec:
  description: AppGroup defines the number of Pods and which Pods belong to the group.
  properties:
    numMembers:
      format: int32
      type: integer
      minimum: 1
      description: Number of Pods belonging to the App Group
    topologySortingAlgorithm:
      type: string
      description: The algorithm for TopologyOrder (Status)
    workloads:
      description: The workloads belonging to the group array of AppGroupWorkload
      items:
        description: AppGroupWorkload contains information about one Workload.
        properties:
          workload:
            properties:
              kind:
                description: Kind is a string value representing the REST resource.
                type: string
              name:
                description: Represents the name of the Object
                type: string
              selector:
                description: Defines how to find pods related to the workload
                type: string
              apiVersion:
                description: APIVersion defines the versioned schema of an object.
                type: string
              namespace:
                description: Represents the namespace of the Object
                type: string
            required:
              - kind
              - name
              - selector
            type: object
          dependencies:
```

App. dependencies with specific affinities

Status

```
status:
  description: Record the number of workload allocations and the favored topology order.
  properties:
    runningWorkloads:
      description: The number of actively running workloads (e.g., pods).
      format: int32
      type: integer
      minimum: 0
    scheduleStartTime:
      description: ScheduleStartTime of the AppGroup
      format: date-time
      type: string
    topologyCalculationTime:
      description: topologyCalculationTime of the AppGroup
      format: date-time
      type: string
    topologyOrder:
      description: The optimal order to schedule workloads on this App Group based on a given algorithm.
      items:
```

Records preferred Topology order

Based on the PodGroup concept introduced for the co-scheduling plugin

NetworkTopology CRD

Spec

```
spec:
  description: NetworkTopology defines the zones and regions of the cluster.
  properties:
    weights:
      description: The weights of the cluster (WeightList)
      items:
        description: WeightInfo contains information about weights of a given algorithm.
        properties:
          name:
            type: string
            description: Algorithm Name (e.g., UserDefined)
          topologyList:
            description: Define weights based on a given topologyKey (TopologyList)
            items:
              description: TopologyInfo contains information about one region.
              properties:
                topologyKey:
                  type: string
                  description: Topology Key (e.g., "topology.kubernetes.io/region", "topology.kubernetes.io/zone")
                originList:
                  description: OriginList contains an array of OriginInfo objects (OriginList).
                  items:
                    description: OriginInfo contains information about one origin.
                    properties:
                      origin:
                        type: string
                        description: Region Name (Origin)
                      costList:
                        description: CostList contains an array of CostInfo objects.
                        items:
                          description: CostInfo contains information about one region.
                          properties:
                            destination:
                              type: string
                              description: Region name (Destination)
                            bandwidthCapacity:
                              anyOf:
                              - type: integer
                              - type: string
                              description: Bandwidth Capacity between Origin and Destination.
                              pattern: ^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$
                              x-kubernetes-int-or-string: true
                            bandwidthAllocated:
                              anyOf:
                              - type: integer
                              - type: string
                              description: Bandwidth allocated between Origin and Destination.
                              pattern: ^(\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))(([KMGTPE]i)|[numkMGTPE]|([eE](\+|-)?(([0-9]+(\.[0-9]*)?)|(\.[0-9]+))))?$
                              x-kubernetes-int-or-string: true
```

Records costs for zones / regions in Kubernetes clusters

Records bandwidth capacity and the bandwidth allocated between zones / regions

Monitor latency through Netperf

90th Percentile Latency

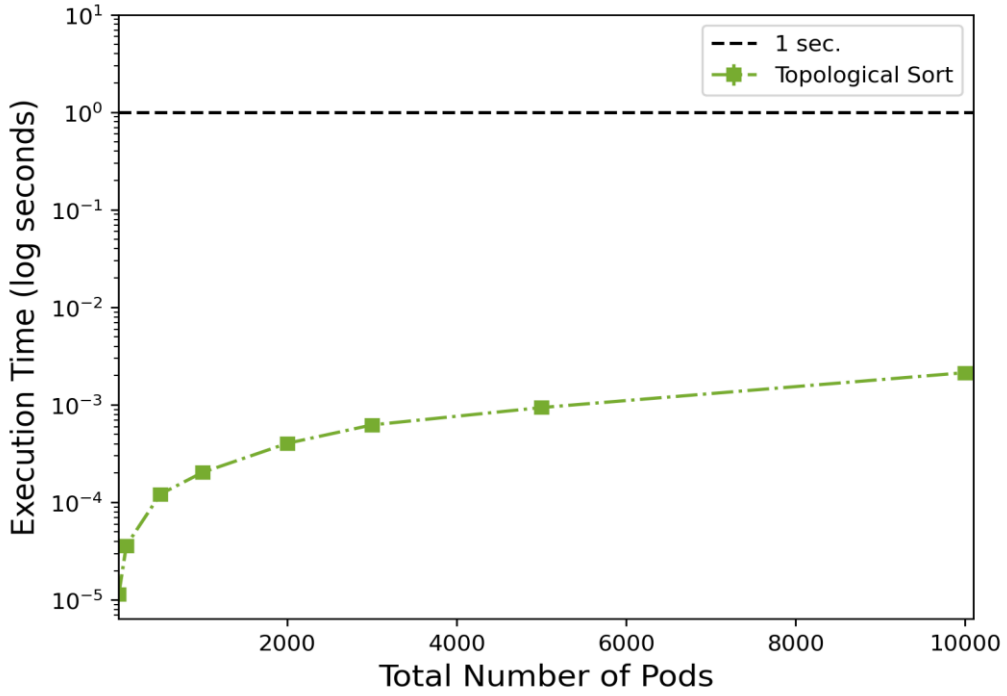
Throughput Units	Throughput	Mean Latency Microseconds	Minimum Latency Microseconds	Maximum Latency Microseconds	50th Percentile Latency Microseconds	90th Percentile Latency Microseconds	99th Percentile Latency Microseconds	Stddev Latency Microseconds	Local CPU Util %
Trans/s	8389.90	118.32	109	582	116	128	170	20.90	5.56
Trans/s	7785.43	127.56	117	418	125	129	148	10.33	5.56
Trans/s	7960.96	124.64	107	1700	120	129	168	61.37	17.48
Trans/s	8351.08	118.89	110	587	115	124	166	23.60	14.17
Trans/s	7749.65	128.08	117	414	125	131	173	11.68	5.00
Trans/s	8082.83	122.80	109	1493	118	129	180	51.49	14.17
Trans/s	7150.57	138.76	111	4553	128	146	190	170.64	17.69
Trans/s	8351.22	118.93	103	1714	114	123	170	63.30	13.64

Run netperf tests and save measurements in configmap

The controller calculates the network costs based on the measured values

Update the Network Topology CR

TopologicalSort (QueueSort) - Performance



The TopologicalSort plugin (QueueSort) increases logarithmically over the number of pods in the queue.

Binary search (O(log n)) is applied to retrieve the pod's topology index.

NetworkOverhead – Filter

- This plugin currently focuses on maxNetworkCost requirements.
- Nodes providing higher network costs than the maxNetworkCost requirement must be filtered out.
- Network weights are available in the Network Topology CR. Nodes that unmet a higher number of dependencies are filtered out to reduce the number of nodes being scored.
- Also, minBandwidth requirements will be considered at a later stage.

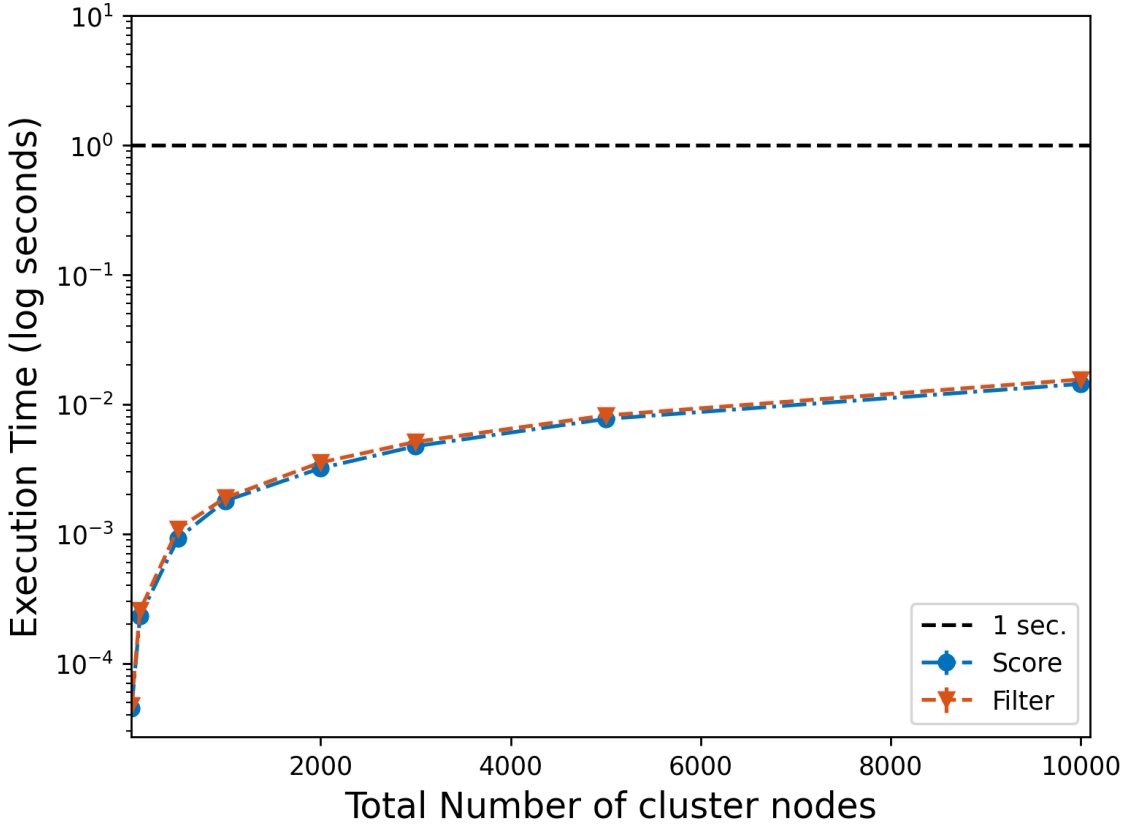
NetworkOverhead – Score

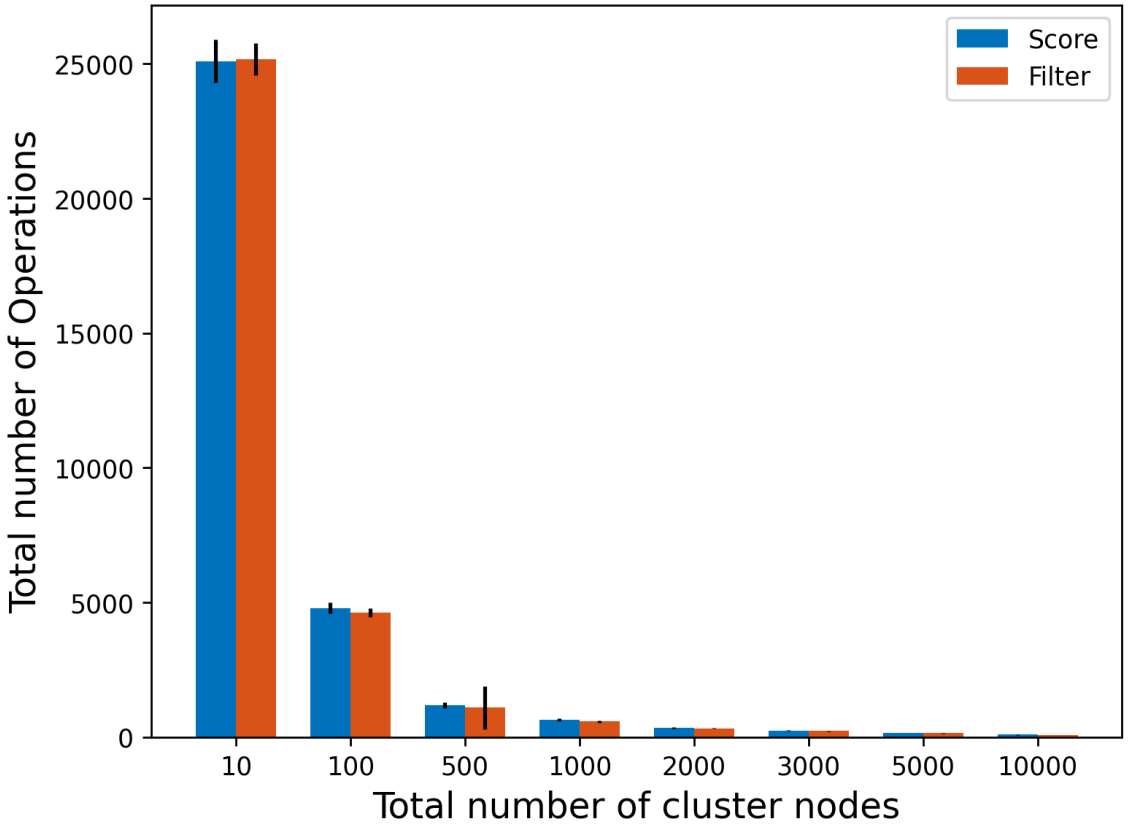
- It favors nodes with the lowest combined cost based on its AppGroup.
- Workload allocations are retrieved via an AppGroup lister.
- Network weights among regions and zones in the cluster are available in the Network Topology CRD.

1. We check if pods are already running for the Pod's AppGroup. Otherwise, we score all candidate nodes equally.
2. We retrieve network costs related to the node being scored.
3. We score the node based on the accumulated network cost related to the workload's dependencies defined in the AppGroup CR. The accumulated cost is returned as the score
4. We get the maximum and minimum costs for all candidate nodes to normalize the values between 0 and 100.

   After normalization, nodes with lower costs are favored since it also corresponds to lower latency.

NetworkOverhead – Performance





Logarithmic performance over the number of nodes well below 1 sec for 10000 cluster nodes.

Implementation Status & Next Steps

KubeCon CloudNativeCon Europe 2022

2021

- September 9th — Presentation to the Kubernetes sig-scheduling community. The framework raised the interest from the community. Received feedback on the design.
- November 4th — Initial Kubernetes Enhancement Proposal (KEP) submitted for revision.

2022

- February 3rd — KEP v0.1 merged in the Kubernetes sig-scheduling repository.
- February 28th — First commit submitted. Waiting for revision.
- March 8th — KubeCon Europe 2022 Talk accepted to showcase the network-aware framework to the CNCF ecosystem.
- Looking for contributors and engagement from the community! ☺

Kubernetes sig-scheduling: https://github.com/kubernetes-sigs/scheduler-plugins
KEP: https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/kep/260-network-aware-scheduling
Initial PR: https://github.com/kubernetes-sigs/scheduler-plugins/pull/348