


KubeCon | CloudNativeCon
North America 2024

Distributed Cache Empowers AI/ML Workloads on Kubernetes Cluster

Yuichiro Ueno, Toru Komatsu (Preferred Networks, Inc.)
KubeCon North America 2024

Speakers

@y1r96

Yuichiro Ueno
Preferred Networks, Inc.

@utam0k

Toru Komatsu
Preferred Networks, Inc.

Today's topic

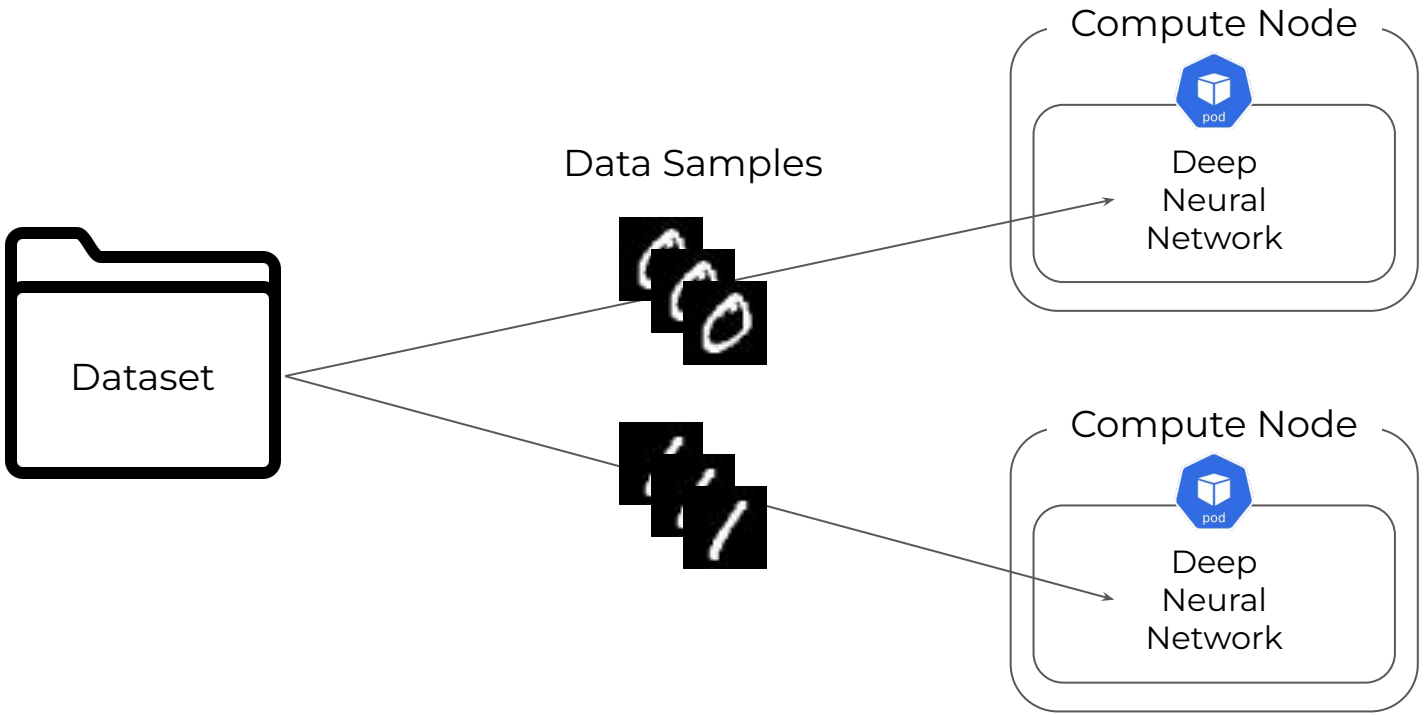
1. **Background: AI / ML Workloads**
   - ✓ Storage Requirements and Kubernetes Usage
2. **Our system: Simple Cache Service**
3. **Use case**
4. **Deploy Considerations**
   - ✓ How to optimize network traffic and key distribution to achieve higher performance
   - ✓ The number of SCS in production
5. **Summary**


KubeCon | CloudNativeCon
North America 2024

Distributed Cache Empowers **AI/ML Workloads** on Kubernetes Cluster

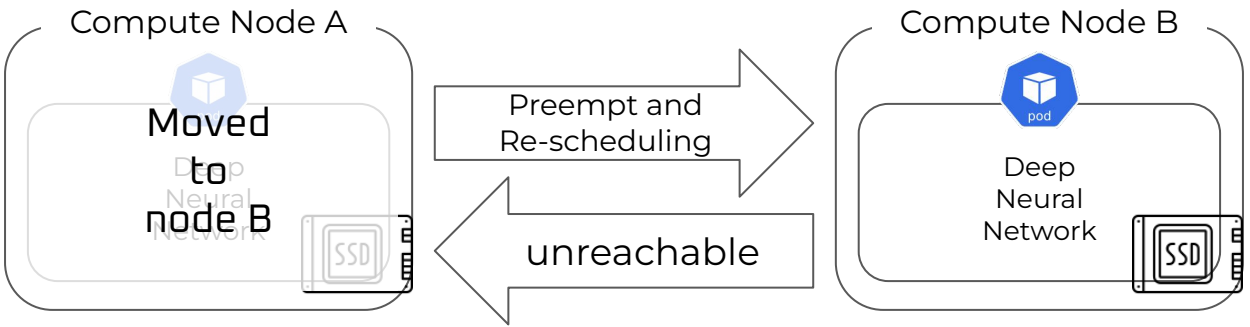
Yuichiro Ueno, Toru Komatsu (Preferred Networks, Inc.)
KubeCon North America 2024

Training of Machine Learning Models

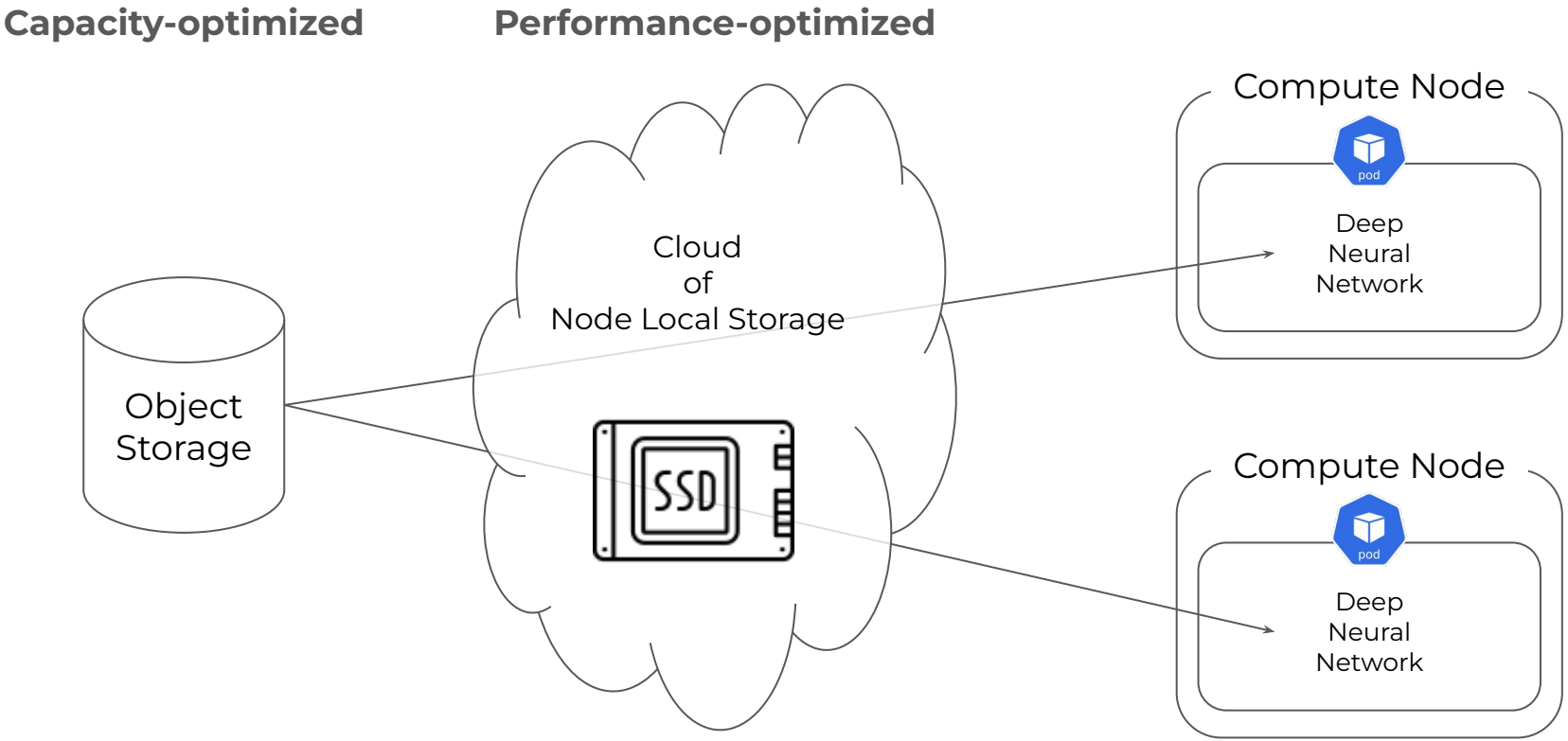


On-premise Storages for the dataset loading

- **Network File System (NFS) with hostPath**
  - Fast but not scalable
- **Object Storage**
  - Scalable but not fast
    - We're using HDDs as backend of our object storage
- **Node Local Storage (NVMe)**
  - Very fast but the storage is not globally available, and not scalable
    - If the workload is moved to different compute node, the data is unreachable.



Best hierarchical storage for AI/ML workload ?



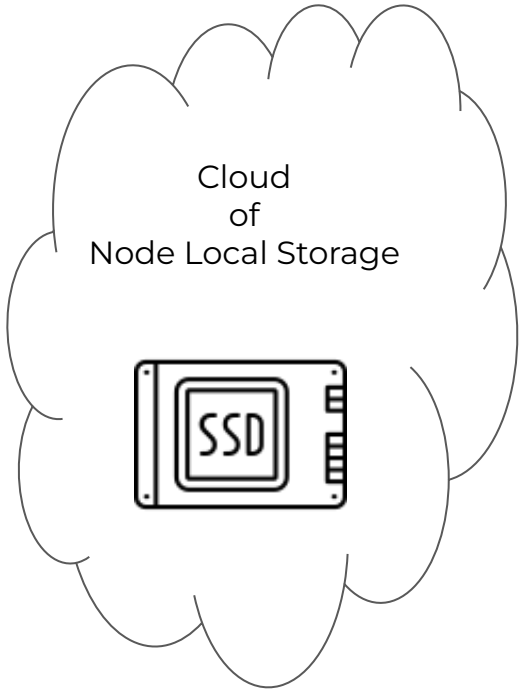
Best hierarchical storage development with:

- ✓ Topology-Aware Routing
- ✓ Informer for Pod Discovery
- ✓ Token Review API

+

- ✓ Consistent Hashing
- ✓ xDS API

=



Overview of Simple Cache Service

Overview of Simple Cache Service

Simple

- ✓ Simple HTTP REST API (GET & PUT)
- ✓ It just returns local files

Cloud Native

- ✓ SCS runs on Kubernetes

Shared-nothing architecture

- ✓ Scalable

Position as a Cache

- ✓ It's just "Cache" and not "Persistent Storage"

How to use SCS

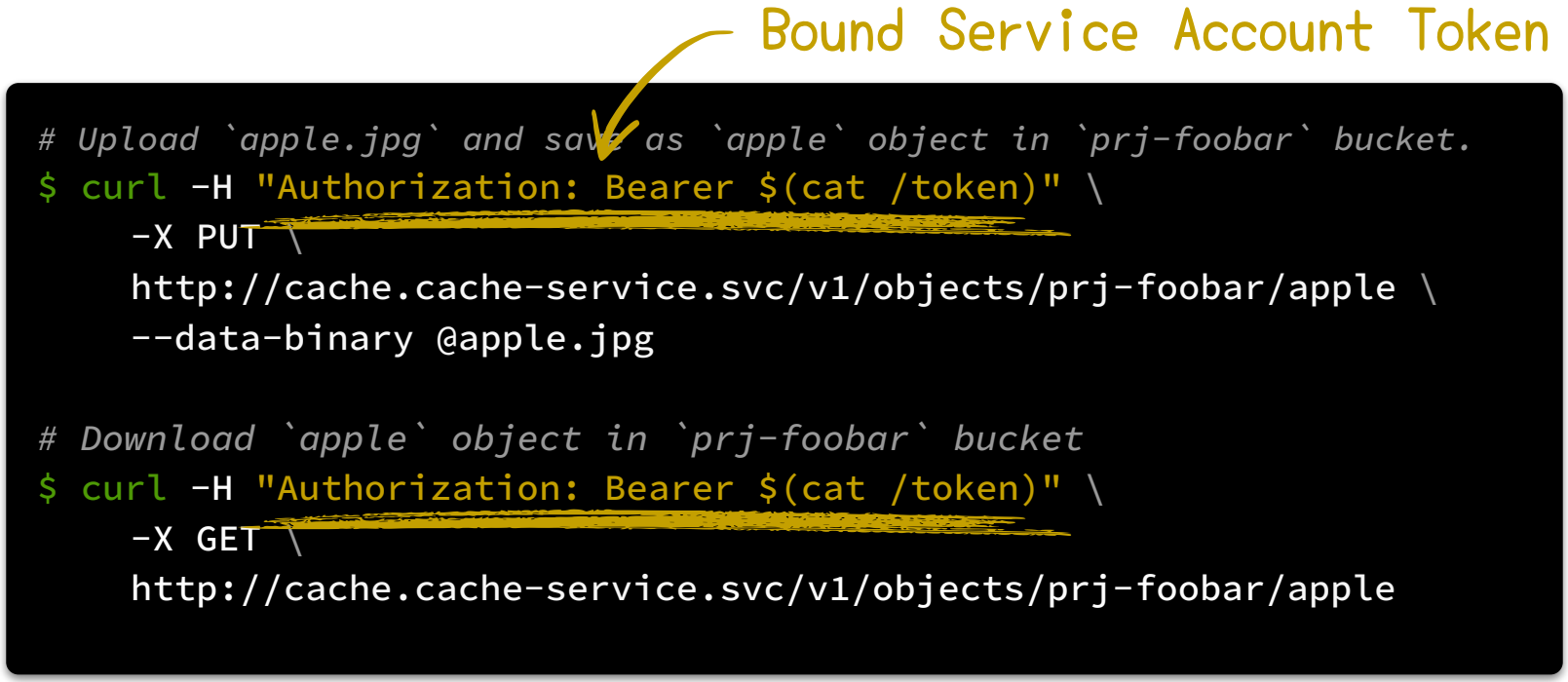

```
# Upload `apple.jpg` and save as `apple` object in `prj-foobar` bucket.
$ curl -H "Authorization: Bearer $(cat /token)" \
    -X PUT \
    http://cache.cache-service.svc/v1/objects/prj-foobar/apple \
    --data-binary @apple.jpg

# Download `apple` object in `prj-foobar` bucket
$ curl -H "Authorization: Bearer $(cat /token)" \
    -X GET \
    http://cache.cache-service.svc/v1/objects/prj-foobar/apple
```

How to use SCS

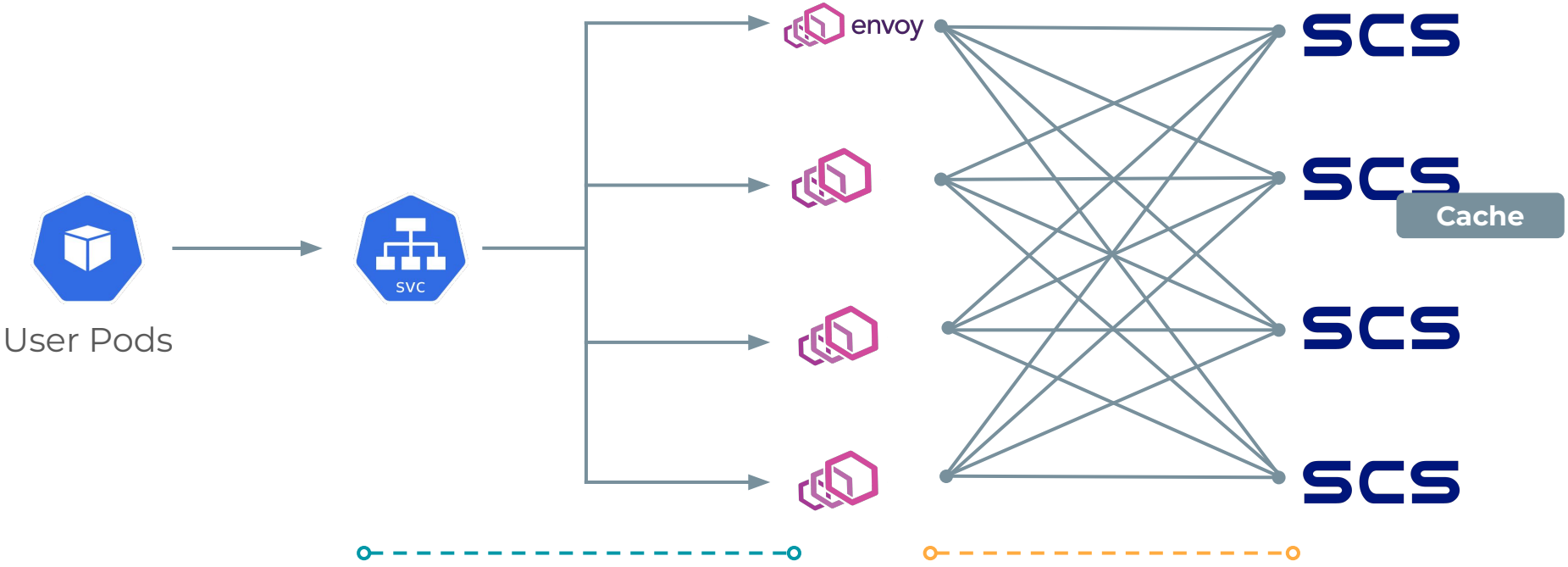

```
# Upload `apple.jpg` and save as `apple` object in `prj-foobar` bucket.
$ curl -H "Authorization: Bearer $(cat /token)" \
    -X PUT \
    http://cache.cache-service.svc/v1/objects/prj-foobar/apple \
    --data-binary @apple.jpg

# Download `apple` object in `prj-foobar` bucket
$ curl -H "Authorization: Bearer $(cat /token)" \
    -X GET \
    http://cache.cache-service.svc/v1/objects/prj-foobar/apple
```

bucket object

How to use SCS



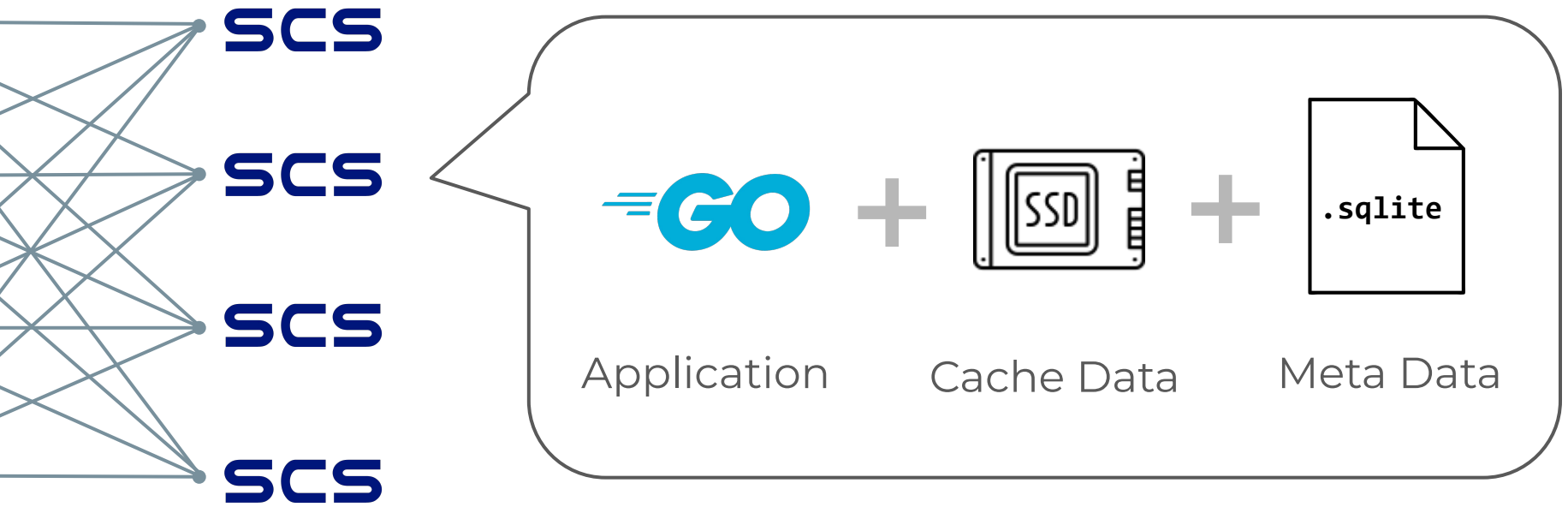
Overall Architecture (1/2)



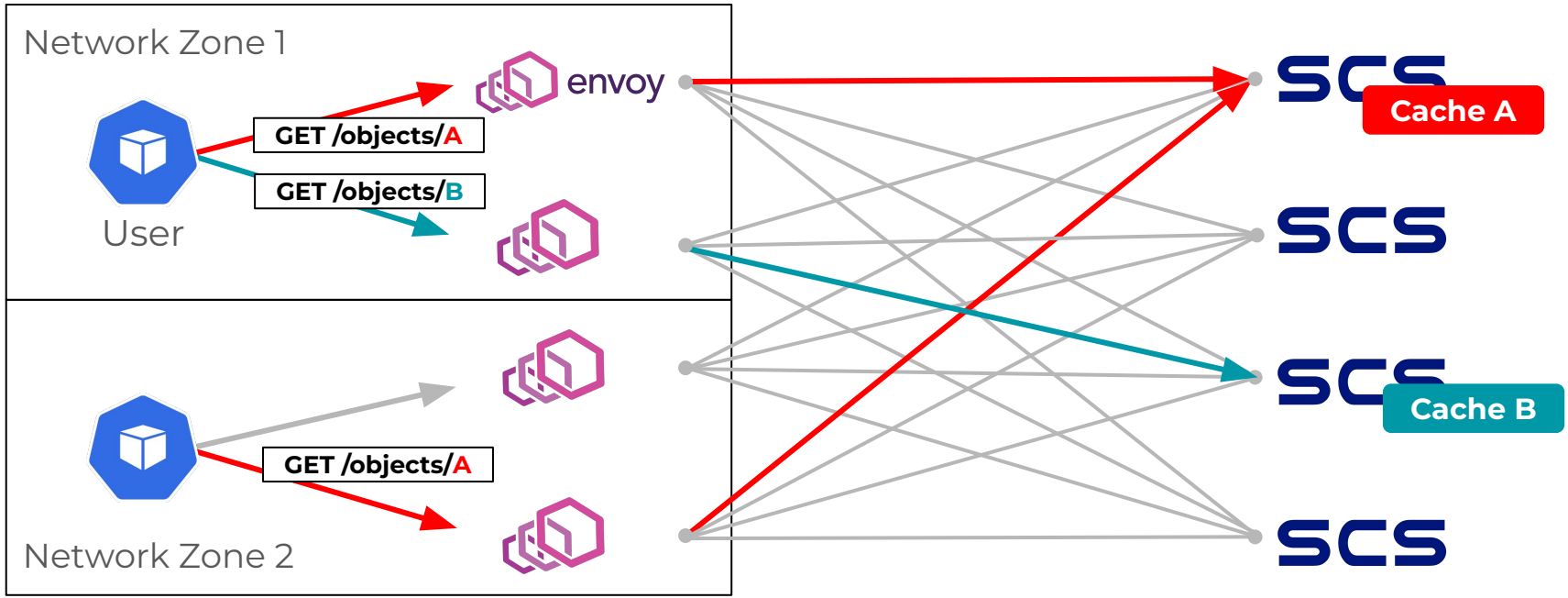
Load Balancing in Layer 4
Service with Topology Aware Hints

Load Balancing in Layer 7
Envoy Proxy with Consistent Hashing

Overall Architecture (2/2)



Shared-nothing Architecture

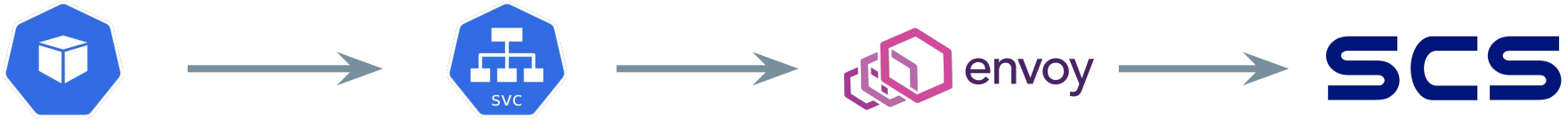


Load Balancing in Layer 4
Service with Topology Aware Hints

Load Balancing in Layer 7
Envoy Proxy with Consistent Hashing

Authorization (1/2)

1. Mount the Bound SA Token
2. Make the request w/ the token in Auth Header



User Pods

3. Verify the token by TokenReview API
   - ✓ “Aud as expected?” “Valid until?” “Pod is still alive?”
   - ✓ Resolve the NS of the source from the SA username
     - ➡ **Namespace-level authorization** can be implemented

Authorization (2/2)

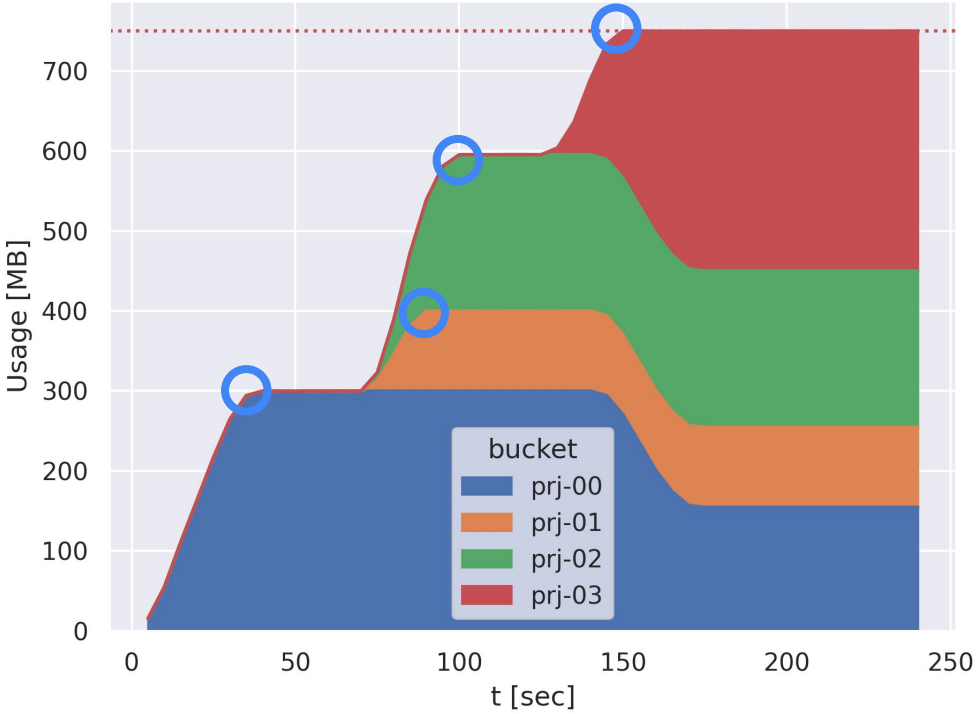

```
"Bucket": [
    {
        "Name": "public",
        "Public": true
        "BucketQuota": "100Gi"
    },
    {
        "Name": "kubecon",
        "Public": false,
        "BucketQuota": "500Gi",
        "AllowNamespaces" : [
            "prj-kubernetes",
            "user-utam0k",
        ]
    }
]
```

Public Bucket

Private Bucket
Based on Namespace Selector

LRU(Least Recently Used) Strategy

Unfortunately, storage is a limited resource... 😭



Total Limit

- When each bucket reaches its capacity limit, object deletion begins based on LRU

Use case

Use case of SCS in AI / ML Workloads

Case 1 SCS as a Cache for Slower Storage

- ✓ Make faster AI/ML Workloads !

Case 2 SCS as a Backend for Yet Another Cache

- ✓ Make faster startup of AI/ML Workloads !

Use case of SCS in AI / ML Workloads

➔ **Case 1** SCS as a Cache for Slower Storage

- ✓ Make faster AI/ML Workloads !

Case 2 SCS as a Backend for Yet Another Cache

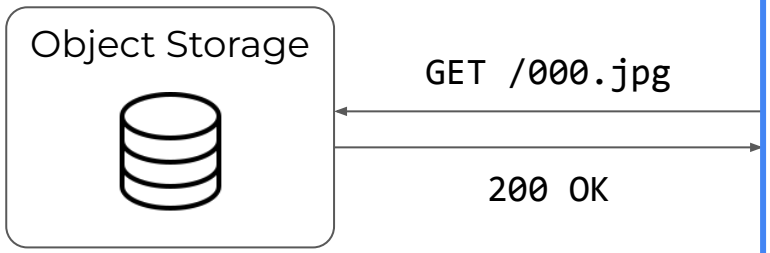
- ✓ Make faster startup of AI/ML Workloads !

Read File in Object Storage with PFIO

Available at: https://github.com/pfnet/pfio

PFIO is an I/O abstraction library developed by us

- It can read / write / list Local filesystem, S3 compatible object storage, HDFS, ...



train_with_scs.py

```python
import pfio
import torch

fs = pfio.v2.from_url(zip_url)
# fs is Local filesystem like object, actually S3

file_url = "000.jpg"
with fs.open(file_url) as fp:
    image = fp.read()

# image = torch.Tensor(image)...
# loss = model(image)
```


Icon pack by Icons8 - https://icons8.com

Transparent Object Storage Cache by PFIO

Available at: https://github.com/pfnet/pfio

PFIO supports transparent cache mechanism

- It automatically checks data in SCS first, then try origin later if data is not exist
- At first, the desired data is not stored in SCS, therefore PFIO will put it to SCS

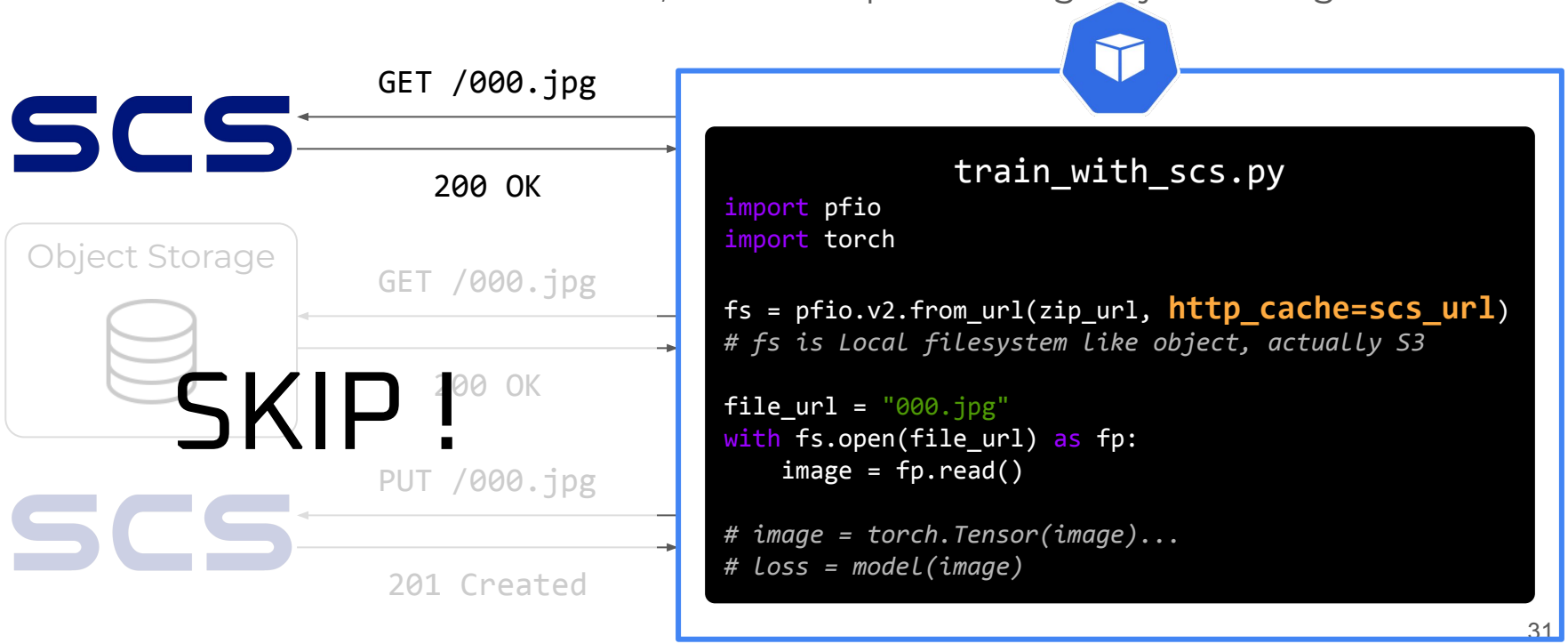
Icon pack by Icons8 - https://icons8.com

Transparent Object Storage Cache by PFIO

Available at: https://github.com/pfnet/pfio

PFIO supports transparent cache mechanism

- It automatically checks data in SCS first, then try origin later if data is not exist
- If the desired data is stored in SCS, we can skip accessing Object Storage



Icon pack by Icons8 - https://icons8.com

Use case of SCS in AI / ML Workloads

Case 1 SCS as a Cache for Slower Storage

- ✓ Make faster AI/ML Workloads !

➔ **Case 2** SCS as a Backend for Yet Another Cache

- ✓ Make faster startup of AI/ML Workloads !

Other large files in AI / ML Workloads

Type 1 Container Images

It includes a lot of dependencies

- Compilers, CUDA (runtime and library), MPI, and PyTorch

As a result, our all-in-one container image is +30 GB

Weekly cache hit rate to SCS is **94.3%** in our cluster

Type 2 Models

Large Language Model is larger and larger !

- GB ~ TB size

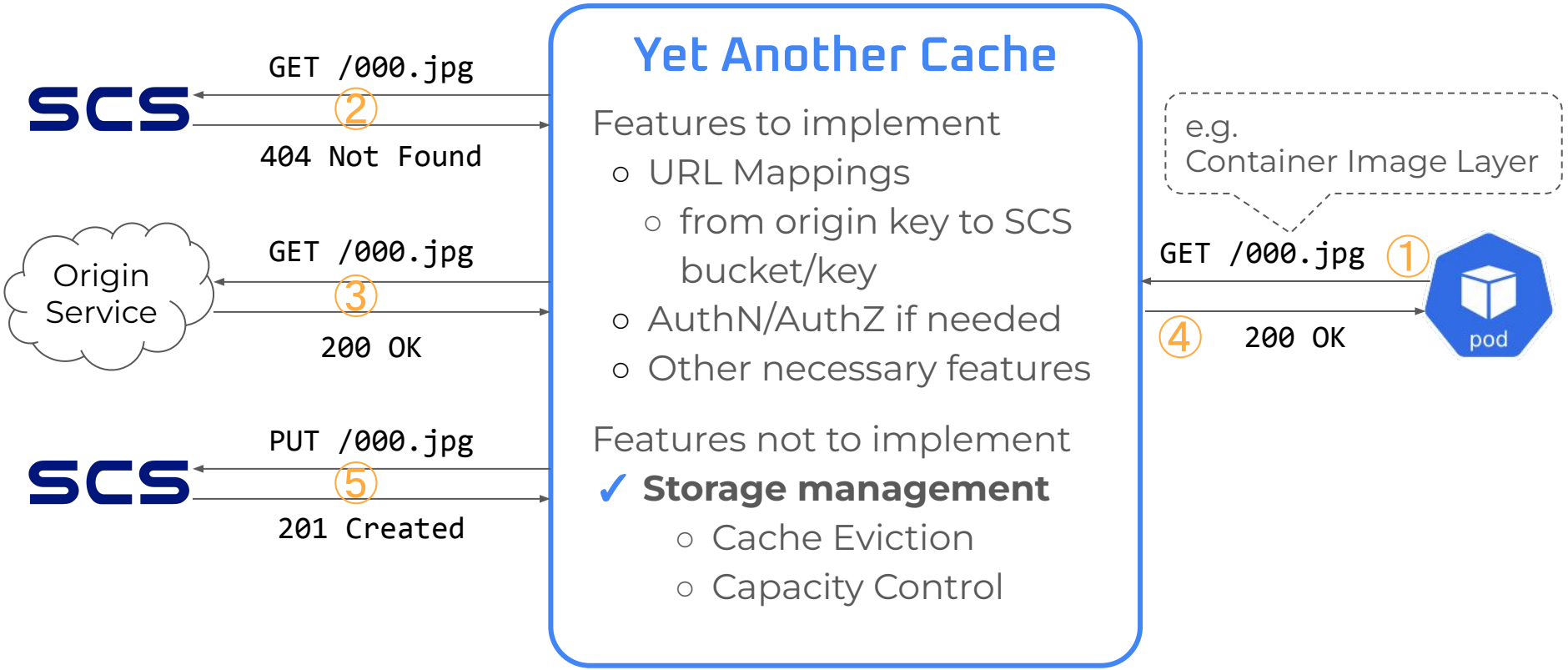
Our researchers want to evaluate the performance of public LLMs

Characteristics Ephemeral, Large, and Hot

Many users access the same file

Cache mechanism works well

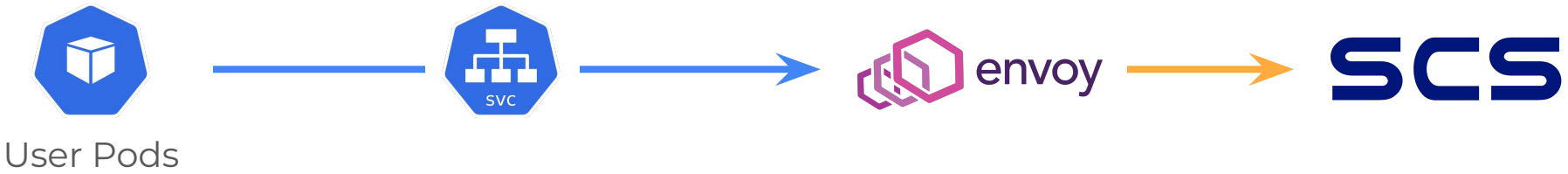
Implementing Yet Another Cache using SCS



Deploying SCS

Deploy Considerations

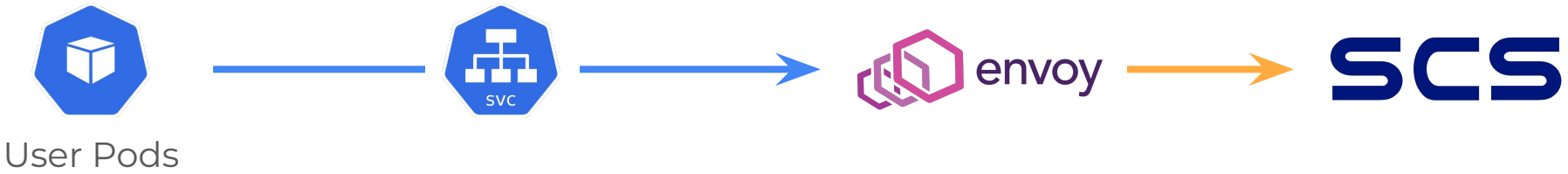
Q1 How can we optimize the network traffic ?



Q2 How can we configure Envoy to route the traffic ?

Deploy Considerations

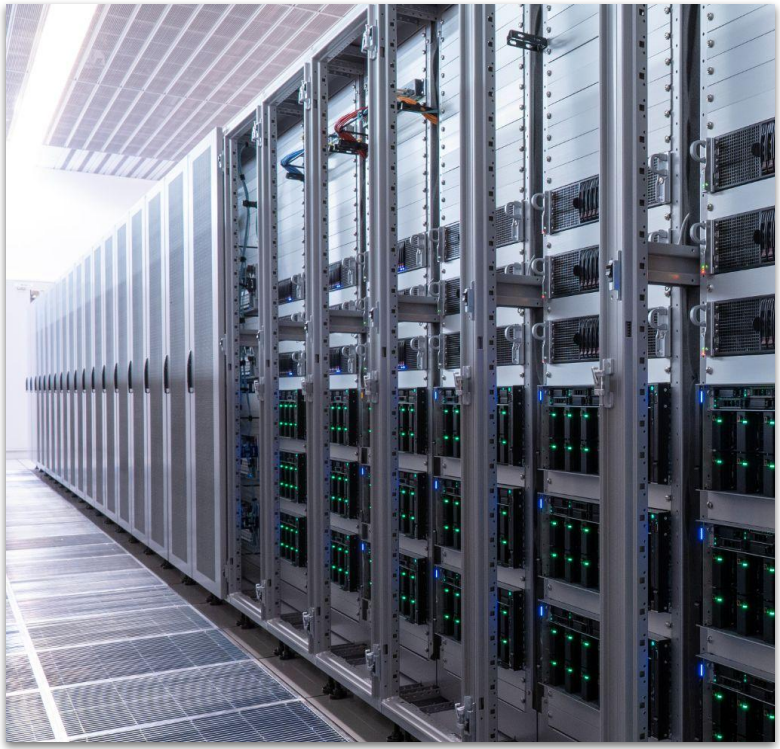
Q1 How can we optimize the network traffic ?



Q2 How can we configure Envoy to route the traffic ?

Background: Our computing infrastructure

Our Infrastructure



Company: Preferred Networks

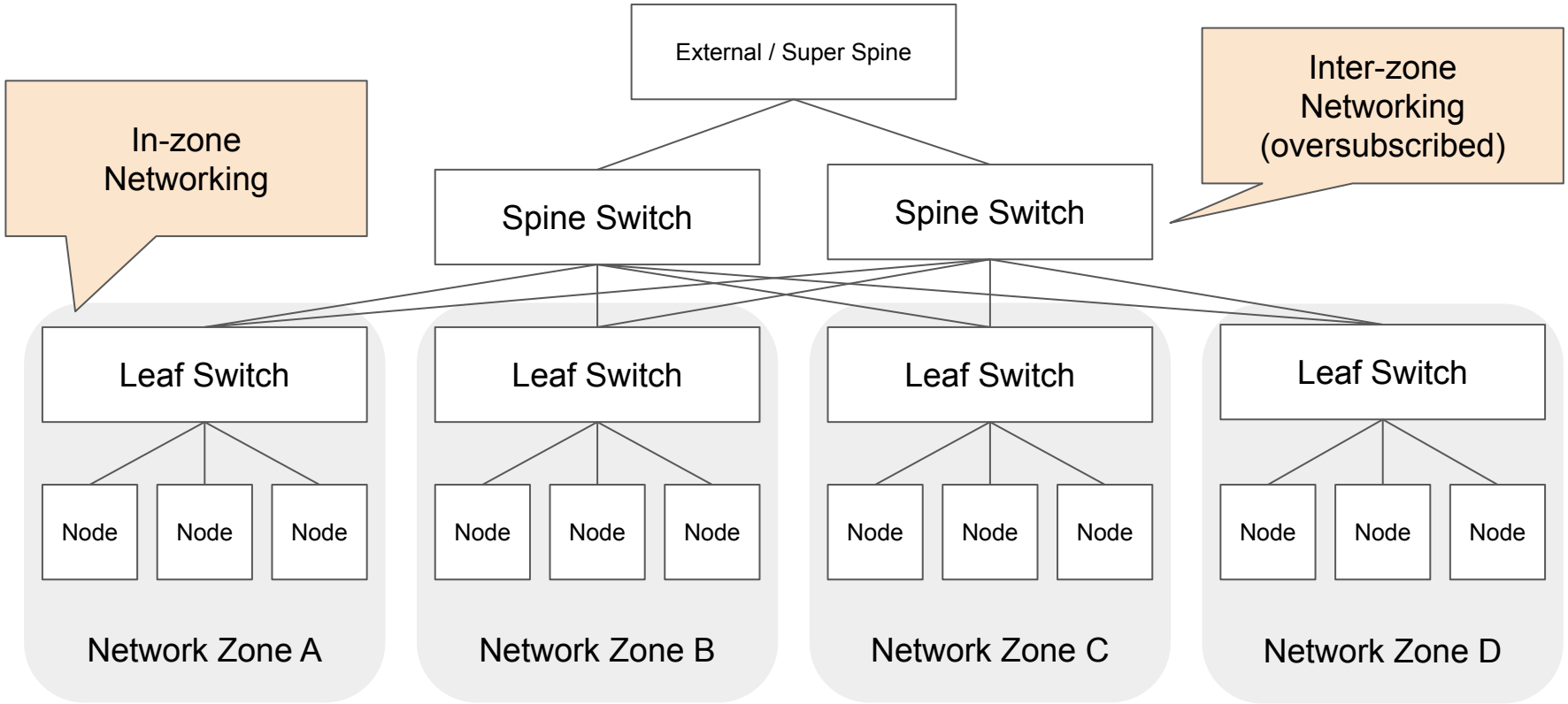
- Provides ML models like LLMs, and solutions for industries
- Uses own on-premise infrastructure to provide solutions

Infrastructure

- 3+ Kubernetes Clusters
- 400+ Kubernetes Nodes
- 30000+ CPU Cores
- 320+ TiB Memory
- 2000+ GPUs
- Our AI Accelerator: MN-Core™
  - HW: RTL, Board/Server Design
  - SW: Driver, Device Plugin, Compiler

Background: Data Center Network

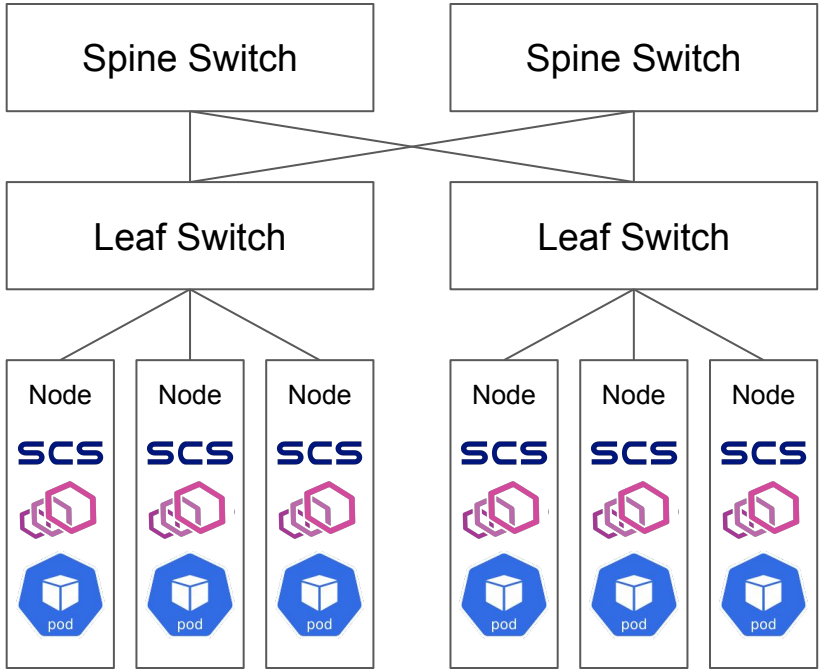
Network topology of our data center: **CLOS network**



Where to deploy Envoy?

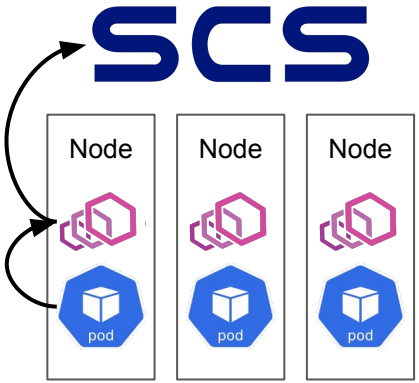
- **Assumptions**
  - SCS is deployed to all nodes to use local NVMe drives
  - Also, User Pods will be scheduled to all nodes to use all accelerators

- **Where to deploy Envoy?**
  - We deploy Envoy to all nodes to reduce inter-zone traffic of Pod/Envoy.
  - Inter-zone traffic of Envoy/SCS is unavoidable in that case.



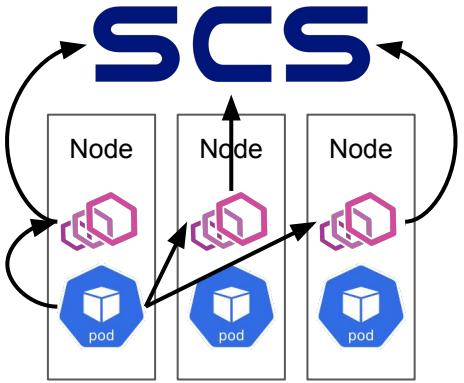
Reducing inter-zone traffic by K8s Service

Internal Traffic Policy



- Pod/Envoy Traffic
  - **Perfect** / No network traffic
- Envoy load balance
  - **Bad** / No distribution of traffic
  - When some node use SCS heavily, the Envoy's cpu load become high

Topology Aware Routing

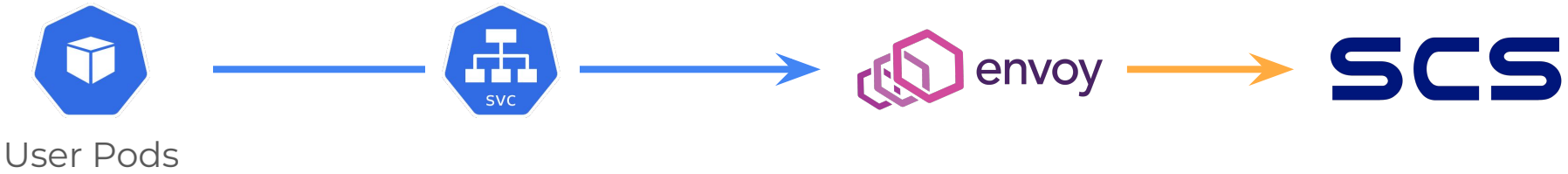


- Pod/Envoy Traffic
  - **Moderate** / In-zone network traffic only
- Envoy load balance
  - **Moderate** / Distribute traffic in a zone
  - When some node use SCS heavily, Envoy's cpu load is distributed among zone

We use **Topology Aware Routing** to improve Envoy's cpu load balance

Deploy Considerations

Q1 How can we optimize the network traffic ?

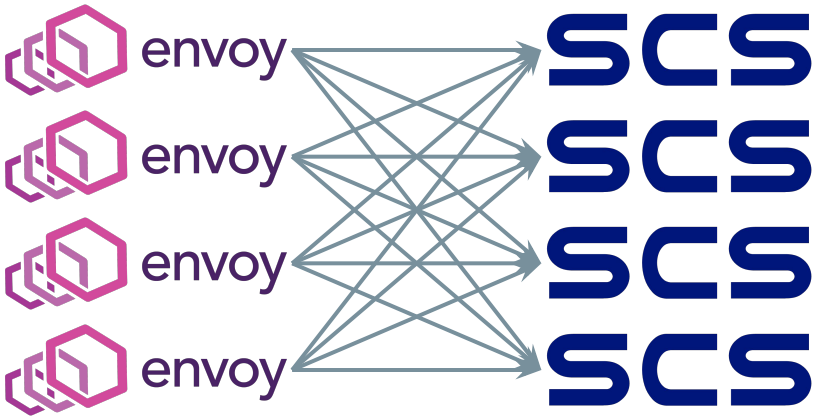


Q2 How can we configure Envoy to route the traffic ?

Load Balancing of Keys (Bucket and Object)

- We want to route the traffic from Envoy to SCS **consistently**
  - *When we put an object to the N-th SCS, we want to get it from the N-th SCS.*

- The easiest way to achieve that:
  - Manage a mapping
    from bucket/object to id of backend
  - Mapping should be sharded...
    - Introduce a distributed MDS
    - Too complicated solution for us



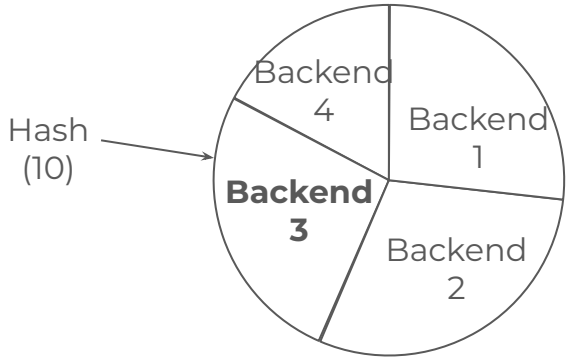
- We don't manage a mapping explicitly
  - Use **hash(bucket + "/" + object)** to choose a backend server

Consistent Hashing

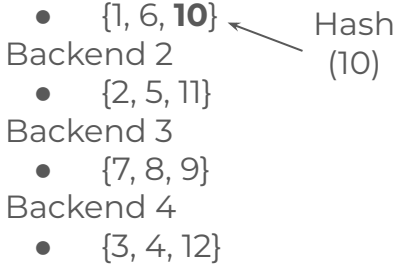
- Use **(hash % number-of-backends)** as backend id ?
  - When the number of backends changes, almost every keys remaps
    - Typical example: Node Failure / Installation
  - More sophisticated way -> Consistent Hashing
    - Bound of remapped keys is keys/backends

- Two Consistent Hashing algorithms in Envoy/lb_policy

RING_HASH

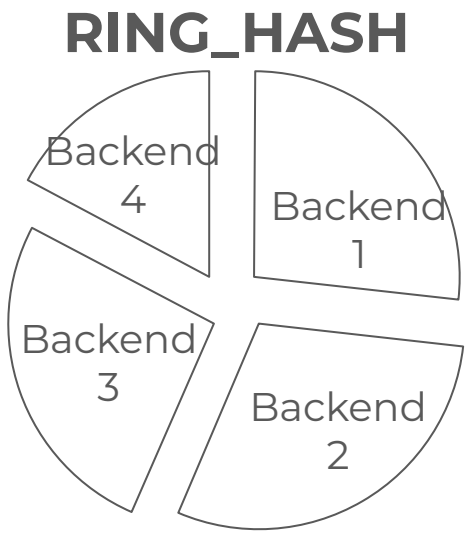


MAGLEV



Key distribution matters!

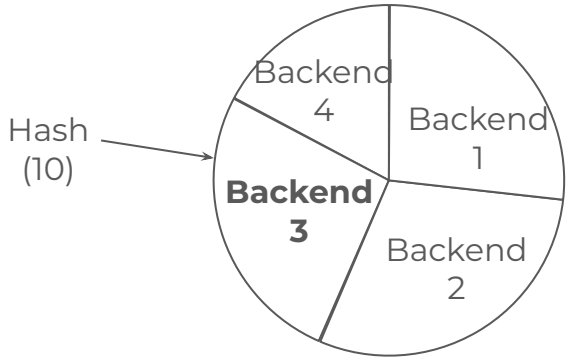
- Load balance of keys is also very important
  - The length of arc in ring_hash corresponds to the ratio of responsibilities
    - **Backend 3 is 1.5x responsibility of Backend 4**
  - It affect the performance !
    - **B3's cpu usage is 1.5x of B4's**
      - Because B3 is 1.5x busier than B4
      - May result the longer latency
    - **The lifetime of B3 data is 1.5x shorter than B4 data**
      - Because the cache capacity is the same
      - More possibility of deletion



We want to see the consistent resource usage and lifetime

RING_HASH vs MAGLEV -> We use MAGLEV

RING_HASH



Load imbalance up to 1.5x

MAGLEV

Backend 1
- {1, 6, **10**}

Backend 2
- {2, 5, 11}

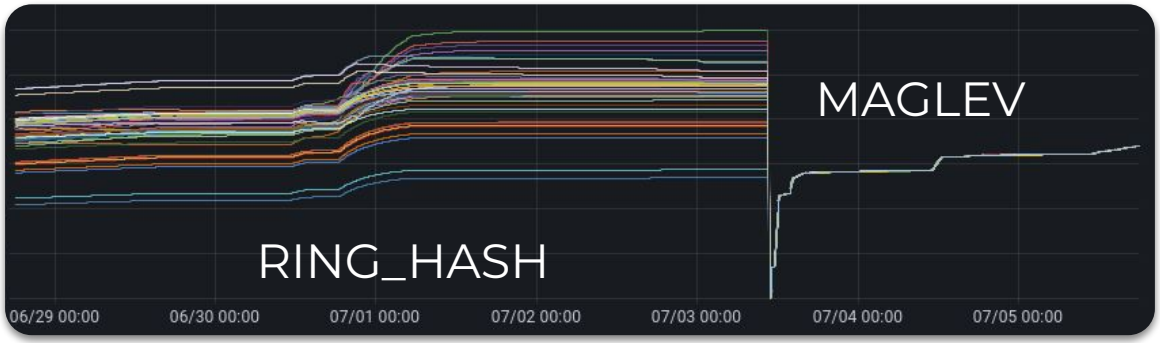
Backend 3
- {7, 8, 9}

Backend 4
- {3, 4, 12}

Hash
(10)

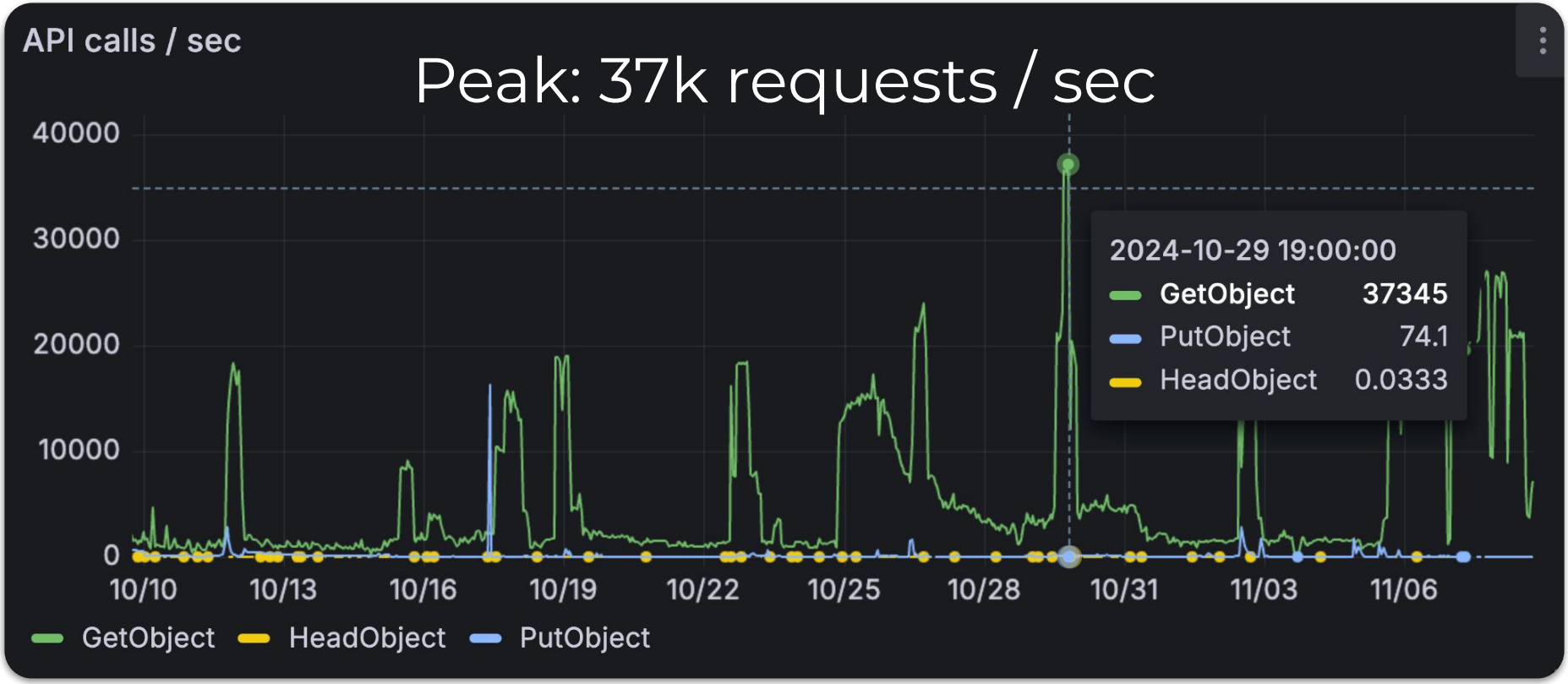
No load imbalance

Objects Count per node

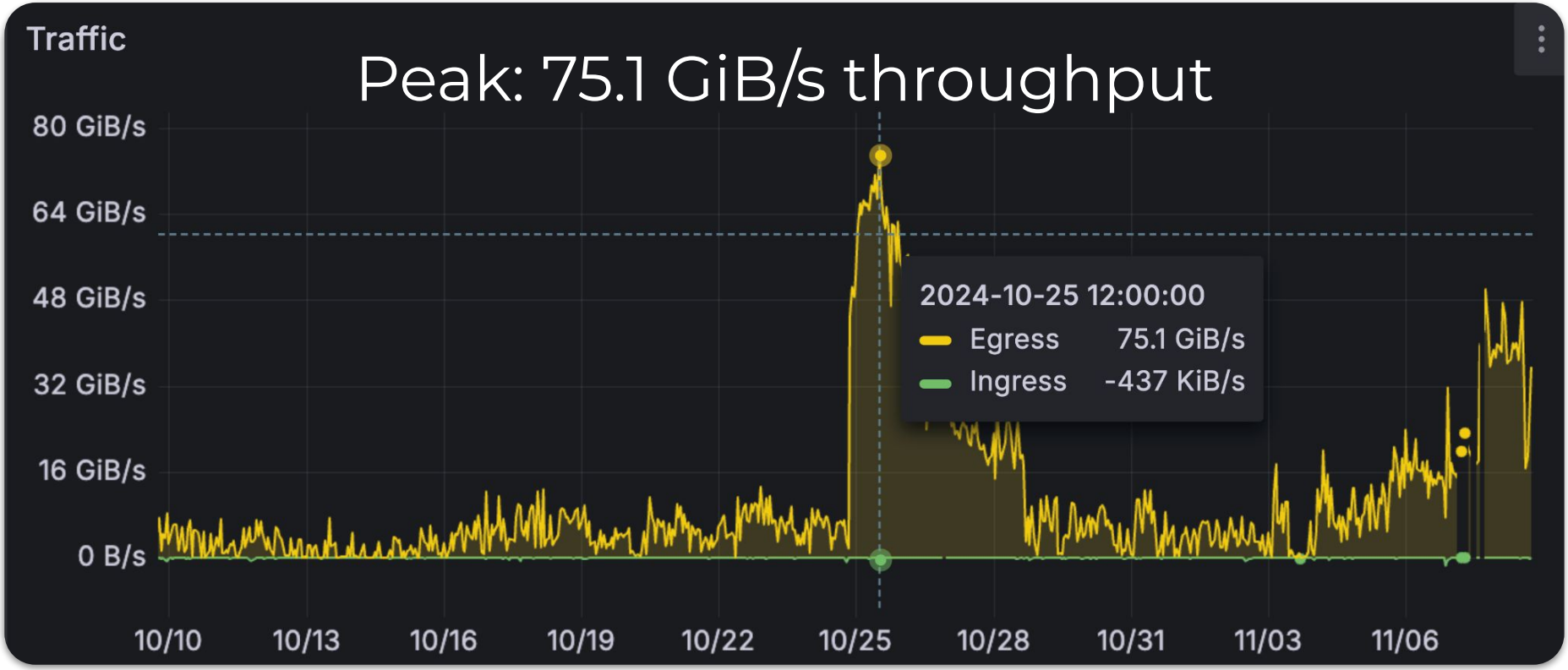


The number of SCS in production

API calls / sec



Numbers of SCS: Aggregated Traffic



Numbers of SCS

- Peak performance of the last 30 days
  - 37k requests / sec
  - 75.1 GiB/s throughput

- We achieved this performance in the production environment with 55 Backend Servers with 82.5 TB NVMe Storage in total

- Usage
  - 268M Objects
  - Response code statistics:
    - 200 OK (GET): 96.2 %
    - 404 Not Found (GET): 0.9 %
    - 201 Created (PUT): 2.9 %

Summary

SCS Summary

Features

- **Feature 1** Shared-nothing: Consistent Hashing with Envoy
- **Feature 2** AuthN / AuthZ: Bound SA Token with TokenReview API
- **Feature 3** Transparent Cache: PFIO

Use cases in the Real World

- **Case 1** AI/ML Dataset Loading
- **Case 2** Large Model Deployment and Container Images

Optimization Techniques

- **Tech 1** CLOS Network optimization: Internal Traffic Policy / Topology-Aware Routing
- **Tech 2** Consistent Hashing Algorithm: RING_HASH / MAGLEV

Supported by

Cloud Native Technologies: Kubernetes and Envoy
Internship members: @naoki9911, @ciffelia, @takonomura

Feedbacks!


KubeCon
CloudNativeCon
North America 2024