Interference-aware Scheduler for Serverless Computing

Jianchang Su, Runxing Wu, Yifan Zhang, Yihang Feng,

Background

Co-location between latency-sensitive services and throughput sensitive jobs is an appealing approach to improving memory utilization in multi-tenant datacenter systems.

Origin of problem: co-location(competition) and unpredictable workload(execution time)

Tradition OS scheduler: CFS not efficient (frequently context switch for short function)

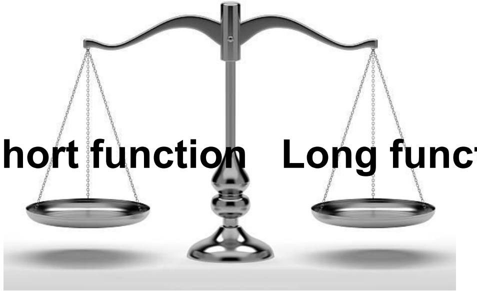
Our goal: develop scheduler to improve Service Level Objectives (SLOs)

Motivation

Solve problem-add another level

CFS - FIFO+CFS

Trade-off



Compute SLO

SLO(service level objectives)

Definition: X% of function invocations are finished within a deadline

How to set: test the function in ideally isolated environment

E.X.: local PC, server

```
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00020432472229003906
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00023865699768066406
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.0002079010009765625
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.0002207756042480468
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.000209808349609375
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.0002155303955078125
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00021338462829589844
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.0002079010009765625
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00020647048950195312
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00020170211791992188
```

```
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test_auth.py
3.0994415283203125e-06
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test_auth.py
3.0994415283203125e-06
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test_auth.py
3.0994415283203125e-06
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test_auth.py
2.6226043701171875e-06
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test_auth.py
2.86102294921875e-06
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test_auth.py
2.6226043701171875e-06
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test_auth.py
2.6226043701171875e-06
```

Influence of function execution time

Different OS: windows 10 vs ubuntu 20.04

```
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00020432472229003906
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00023865699768066406
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.0002079010009765625
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00022077560424804688
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.0002098083496093750
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.0002155303955078125
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00021338462829589844
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.0002079010009765625
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00020647048950195312
root@SFSRunxin:/home/sfswu/Desktop/project# python3 test1.py
0.00020170211791992188
```

```
(base) C:\Users\xitian\Desktop>python test1.py
0.0019485950469970703

(base) C:\Users\xitian\Desktop>python test1.py
0.0009770393371582031

(base) C:\Users\xitian\Desktop>cd Desktop
系统找不到指定的路径。

(base) C:\Users\xitian\Desktop>python test1.py
0.0

(base) C:\Users\xitian\Desktop>python test1.py
0.0019547939300537111

(base) C:\Users\xitian\Desktop>python test1.py
0.0019533634185791016
```

Influence of function execution time

Sever vs ubuntu

Execution time:

windows 10 < ubuntu 22.04

Ubuntu 22.04 < server

conclusion:

Which ideally isolated environment is the best

Keep the environment the same

step

1. Extract the critical function
2. Run the function several time (same environment)
3. Compute the average value of the execution time.
4. Modify this value

```
def generatePolicy(principalId, effect, resource):
    start_time=time.time()
    authResponse = Empty()
    authResponse.principalId = principalId
    if effect and resource:
        policyDocument = Empty()
        policyDocument.Version = '2012-10-17'
        policyDocument.Statement = [None]
        statementOne = Empty()
        statementOne.Action = 'execute-api:Invoke'
        statementOne.Effect = effect
        statementOne.Resource = resource
        policyDocument.Statement[0] = statementOne
        authResponse.policyDocument = policyDocument

    # Optional output with custom properties of the String, Number or Boolean type.
    authResponse.context = {
      "stringKey": "stringval",
      "numberKey": 123,
      "booleanKey": True
    }
    end_time=time.time()
    execute_time=end_time-start_time
    print(execute_time)
    return authResponse
if __name__ == '__main__':
    resource = "arn:aws:execute-api:{regionId}:{accountId}:{apiId}/{stage}/{httpVerb}/[{resource}/[{child-resources
    generatePolicy('user', 'Allow', resource)
```

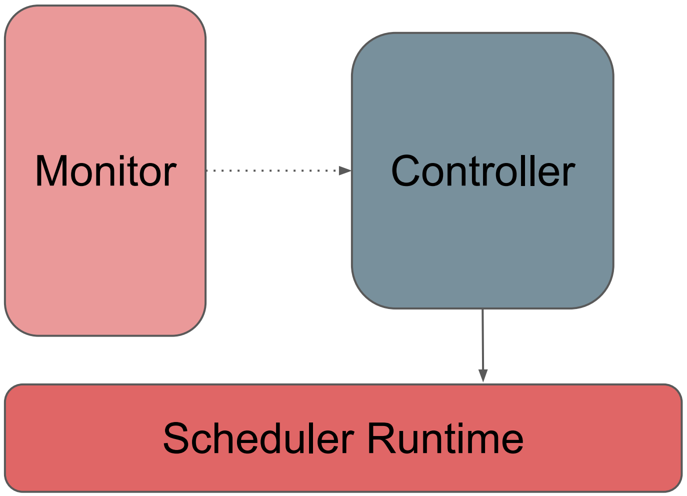
```
import os
import sys
import pyaes
import argparse
import time
parser = argparse.ArgumentParser()
parser.add_argument("--default_plaintext", default="defaultplaintext", help="Default plain text if plaintext_messag
parser.add_argument("-k", "--key", dest="KEY", default="6368616e676520746869732070617373", help="Secret key")
args = parser.parse_args()
KEY = args.KEY.encode(encoding = 'UTF-8')
def AESModeCTR(plaintext):
    start_time=time.time()
    counter = pyaes.Counter(initial_value = 0)
    aes = pyaes.AESModeOfOperationCTR(KEY, counter = counter)
    ciphertext = aes.encrypt(plaintext)
    end_time=time.time()
    execute_time=end_time-start_time
    print(execute_time)
    return ciphertext


if __name__ == '__main__':
    plaintext = args.default_plaintext
    AESModeCTR(plaintext)
```

Framework Implementation - Overview

- Our Framework involves three main components

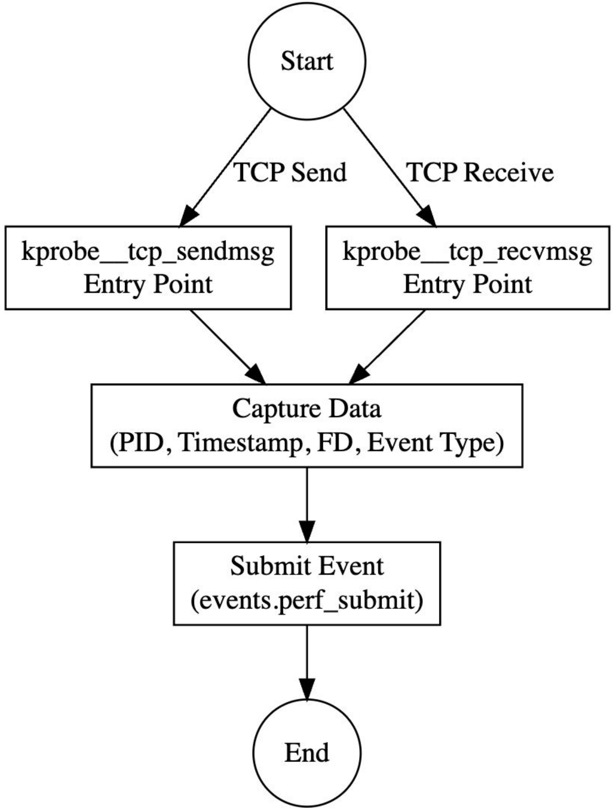
- Scheduler Runtime
- Monitor
- Controller

Framework Implementation - Overview

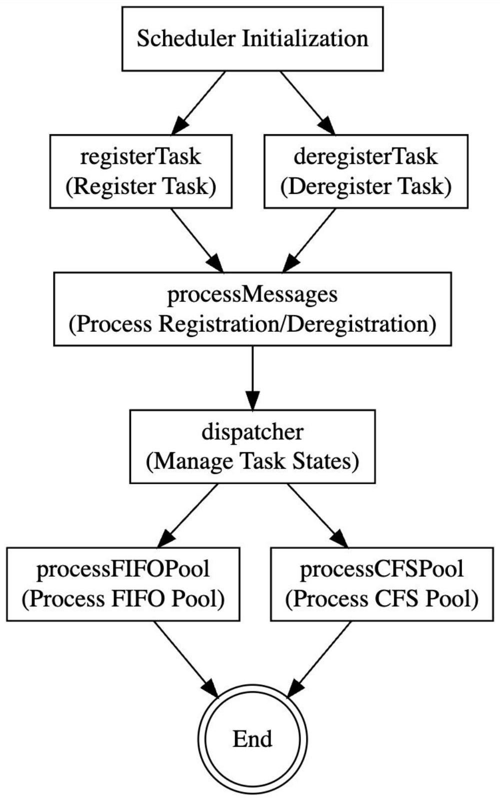


Framework Implementation - Monitor

Specifically, we use kprobes on ***tcp_sendmsg*** and ***tcp_recvmsg*** functions, capturing data such as **process ID (PID)**, **timestamp**, **file descriptor (FD)**, and **event type**

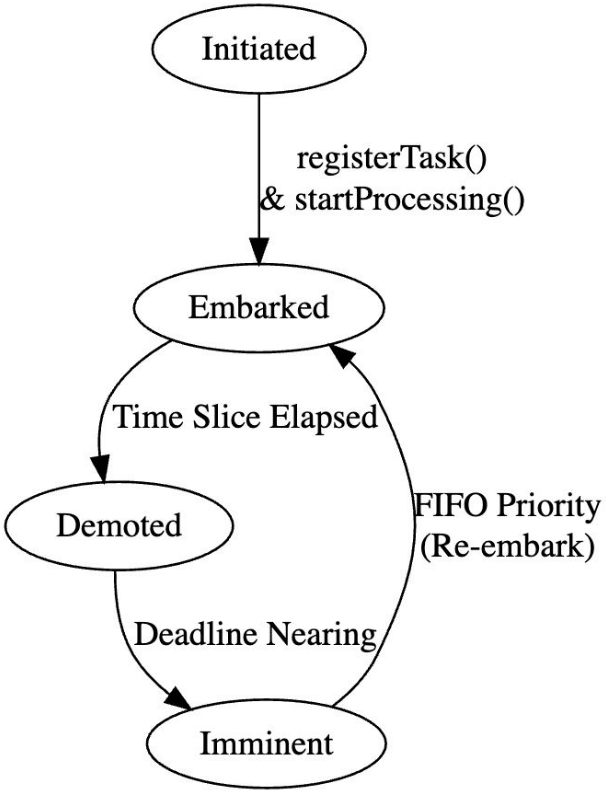


Framework Implementation - Runtime

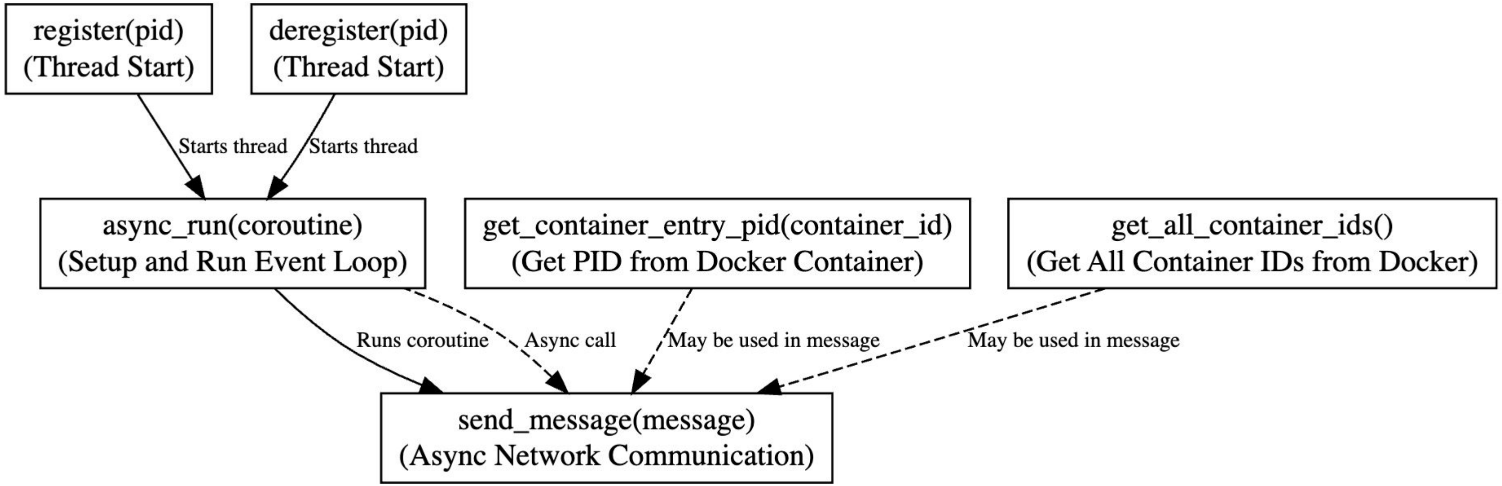


Framework Implementation - Runtime

The tasks transition between various states (**Initiated**, **Embarked**, **Demoted**, **Imminent**).



Framework Implementation - Controller

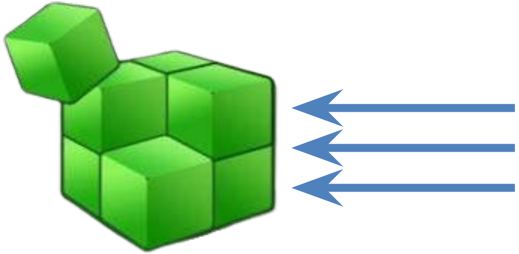


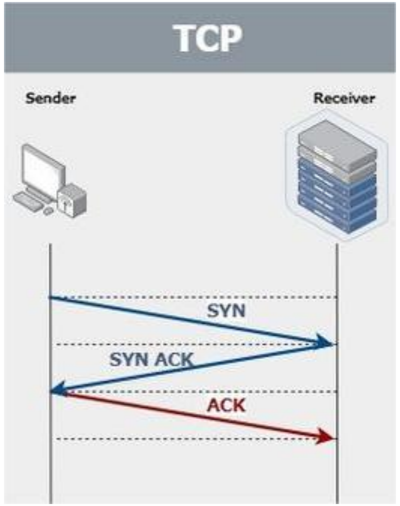
Register

One big challenge in designing and implementing the register is that our platform uses TCP socket to trace the requests. So the register has to handle request status changes over multiple packets.

Example of the trace:

CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: recv, Timestamp: 93642890.399628 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: recv, Timestamp: 93642890.407442 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: send, Timestamp: 93647891.92718 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: send, Timestamp: 93647891.940696 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: send, Timestamp: 93647892.062655 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: send, Timestamp: 93647892.064929 milliseconds





A dictionary variable was used to maintain a register, monitoring the current active requests.

UCONN

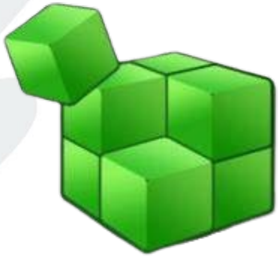
Register

When to register/deregister requests?

```
if FD not in dict and event == recv:
        dict[FD]=request status
if FD in dict and event == send:
        del dict[FD]
```

register

deregister



The register is maintained as a global variable in the monitor program.

CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: recv,
Timestamp: 93642890.399628 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: recv,
Timestamp: 93642890.407442 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: send,
Timestamp: 93647891.92718 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: send,
Timestamp: 93647891.940696 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: send,
Timestamp: 93647892.062655 milliseconds
CPU: 28, PID: 61883, FD: 3843769, Size: 36, Event: send,
Timestamp: 93647892.064929 milliseconds

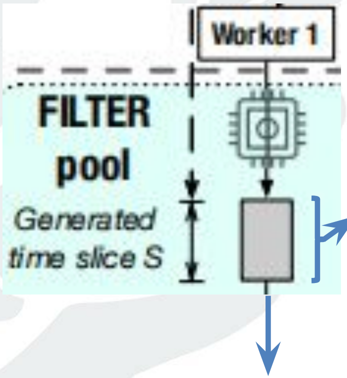
UCONN

Time slice

The inter arrival rate (IAT) was used to dynamically adjust the time slice
S=IAT * c, c is the number of commuter cores.

```
inter arrival time = 1/arrival rate
```

Design of dynamical time slice:
(1)Accommodate the skewed execution duration of the workload.
(2)Strike a balance between queuing delay and execution time.



Adjusted by IAT of last 100 requests.
Initial S = 500 ms

CFS pool

A list variable is maintained to record the last 100 requests in the monitor program.

```
history_requests.append((event.fd,event.timestamp / 1000000))
n = len(history_requests)
if n <= N and n>1:
  IAT = (event.timestamp / 1000000 - history_requests[0][1]) / (n-1)
elif n>N:
  IAT = (event.timestamp / 1000000 - history_requests[0][1]) / N
  history_requests.pop(0)
```

Time slice

$S = IAT * c$

$\rho = \lambda/(c\mu)$, according to the M/G/c model with Kendall's notation in queuing theory.

ρ: traffic intensity per core
λ: arrival rate of the requests
μ: service rate of a single core
c: the number of cores used

$$S \xrightarrow{\text{bound}} \mu \xrightarrow{\text{affect}} \rho \xrightarrow{\text{affect}} \text{queuing delay}$$

Design intuition

All functions whose execution duration is shorter than S run to completion without being preempted.

Balance

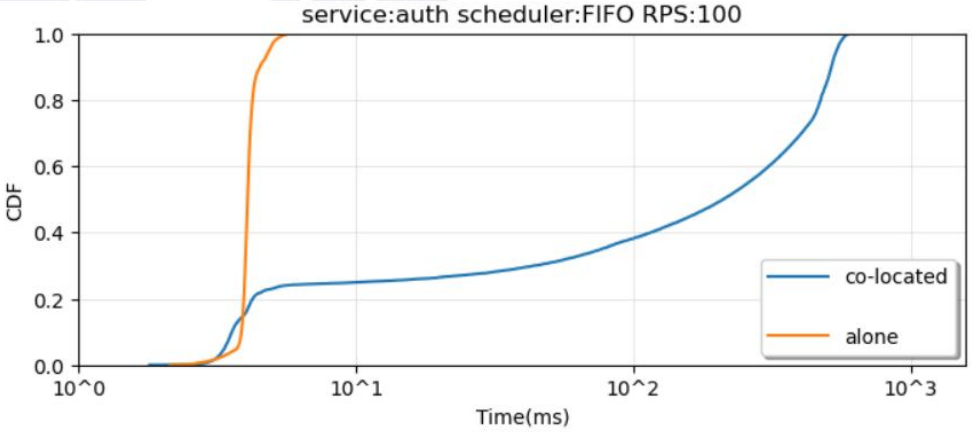
UCONN

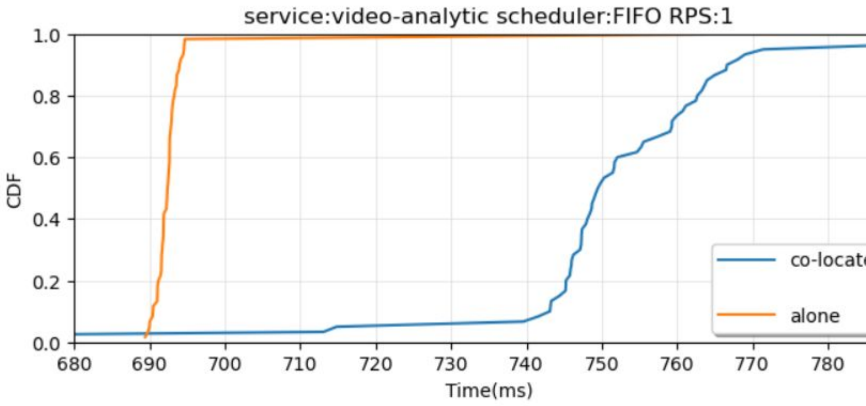
Evaluation:baseline

Baseline

Service: 1.auth 2.video-analytic

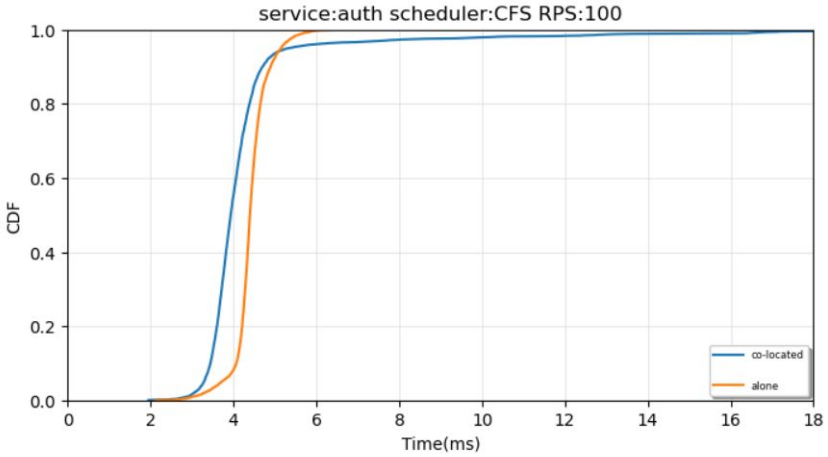
Scheduler: FIFO

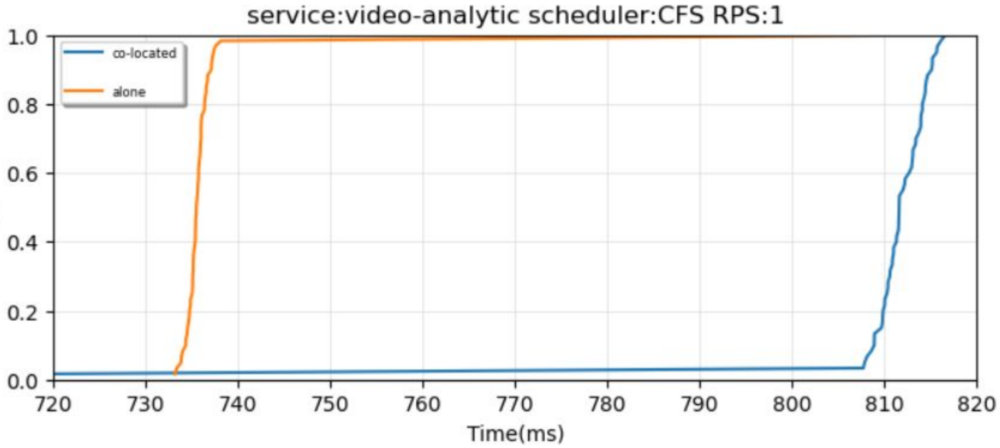




Evaluation:baseline

Baseline
Service: 1.auth 2.video-analytic
Scheduler: CFS





UCONN

Evaluation

evaluation

Service: 1.auth 2.video-analytic

Scheduler: project scheduler