Latency Reduction between Mobile Users and Edge Servers

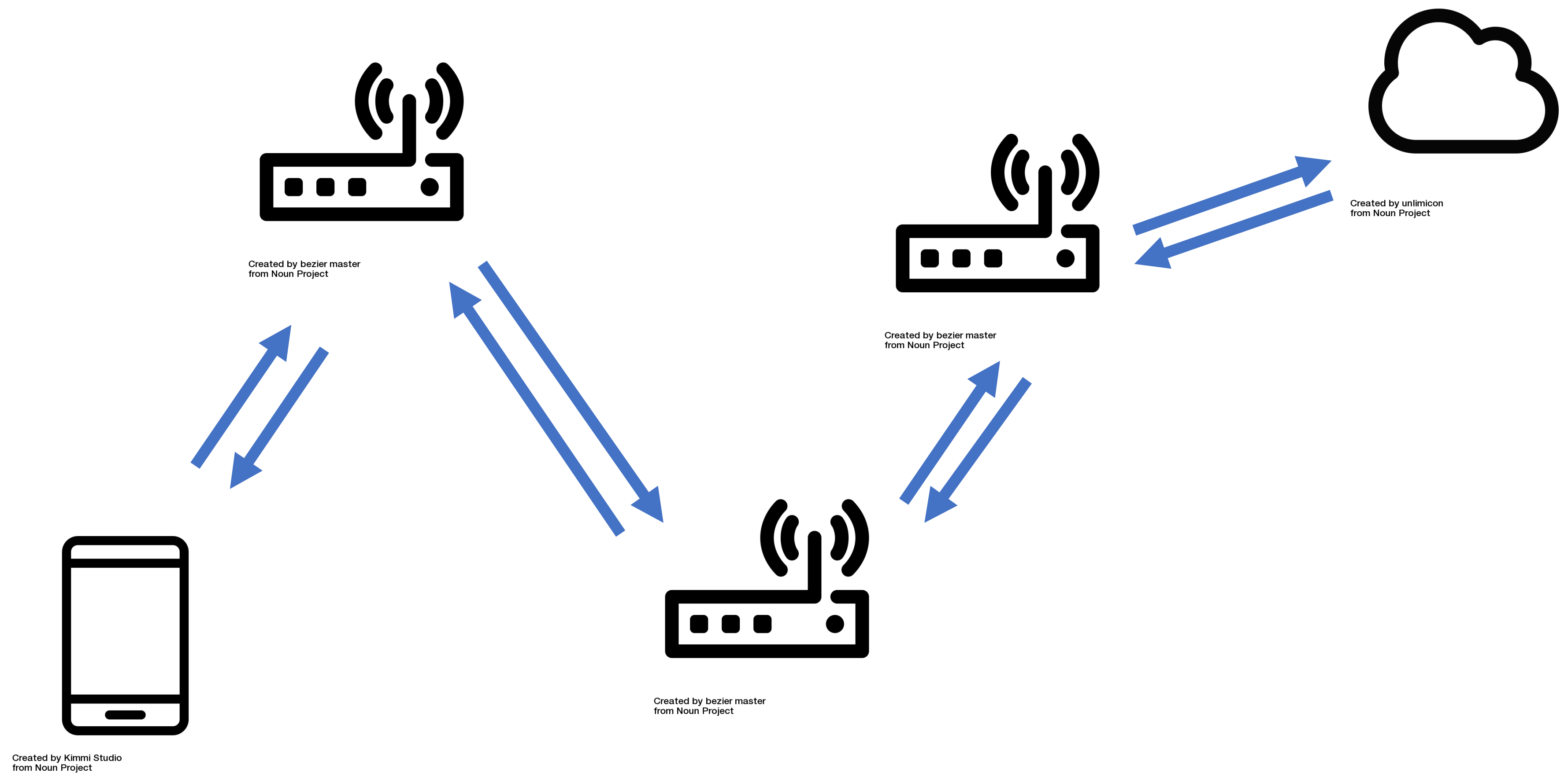
Team Cloudy (구르미조)

Lab:	Mobile Networking Lab
Professor:	Seung-Jae Han
TA:	Sung Woong Jung
Members:	Jae-ho Jung (2011147106) Myung-jong Kim (2011147026) Samuel Shin (2012111008)

1. Research goal: Latency reduction between mobile users and edge servers

- Low processing power
- Limited battery life

- Workloads can be “offloaded” to the cloud
- Optimized for CPU/GPU-intensive workloads


Created by bezier master
from Noun Project
Created by bezier master
from Noun Project
Created by unlimicon
from Noun Project
Created by bezier master
from Noun Project
Created by Kimmi Studio
from Noun Project

Goals:

- Implement techniques to reduce workload offloading latency between mobile devices and cloud servers
- Define appropriate situations for different workload offloading techniques

Techniques:

- Operating-system-level virtualization vs. full virtualization
- Custom scheduling algorithm
- Hop count minimization

2. Related work

COSMOS: Computation Offloading as a Service for Mobile Devices (ACM, 2014.)

- Cloud offloading
  - AWS EC2 instance
  - Android x86 image
- Uses execution predictor for offloading decision
  - Mantis: Automatic performance prediction for smartphone applications (USENIX, 2013.)

Shi, Cong, et al. "Cosmos: computation offloading as a service for mobile devices." *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2014.

Content-Based Scheduling of Virtual Machines (VMs) in the Cloud (IEEE, 2013.)

- Similar VM images have many identical disk blocks
  - Similarity of Ubuntu 12.04 and Ubuntu 11.10 = 60%
  - Minimize VM provisioning overhead when creating multiple VMs
- Schedule workload process to minimize network traffic

Bazarbayev, Sobir, et al. "Content-based scheduling of virtual machines (VMs) in the cloud." *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013.

Just-in-Time Provisioning for Cyber Foraging (ACM, 2013.)

- Reduce cloudlet VM provisioning overhead
  - Deduplication: only transfer overlay information to base VM
  - Identify and ignore “garbage” disk blocks
  - Communicate free memory information between VM and VM manager
  - VM synthesis pipelining

Ha, Kiryong, et al. "Just-in-time provisioning for cyber foraging." *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 2013.

Cloudlet-Based Cyber-Foraging for Mobile Systems in Resource-Constrained Edge Environments (ACM, 2014.)

- Introduce multiple cloudlet provisioning strategies
  - Optimized VM synthesis
  - Application virtualization
  - Cached VM
  - Cloudlet push
  - On-demand VM provisioning

Lewis, Grace A., et al. "Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments." *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014.

3. Implementation

Cloudy

Created by jngll
from Noun Project

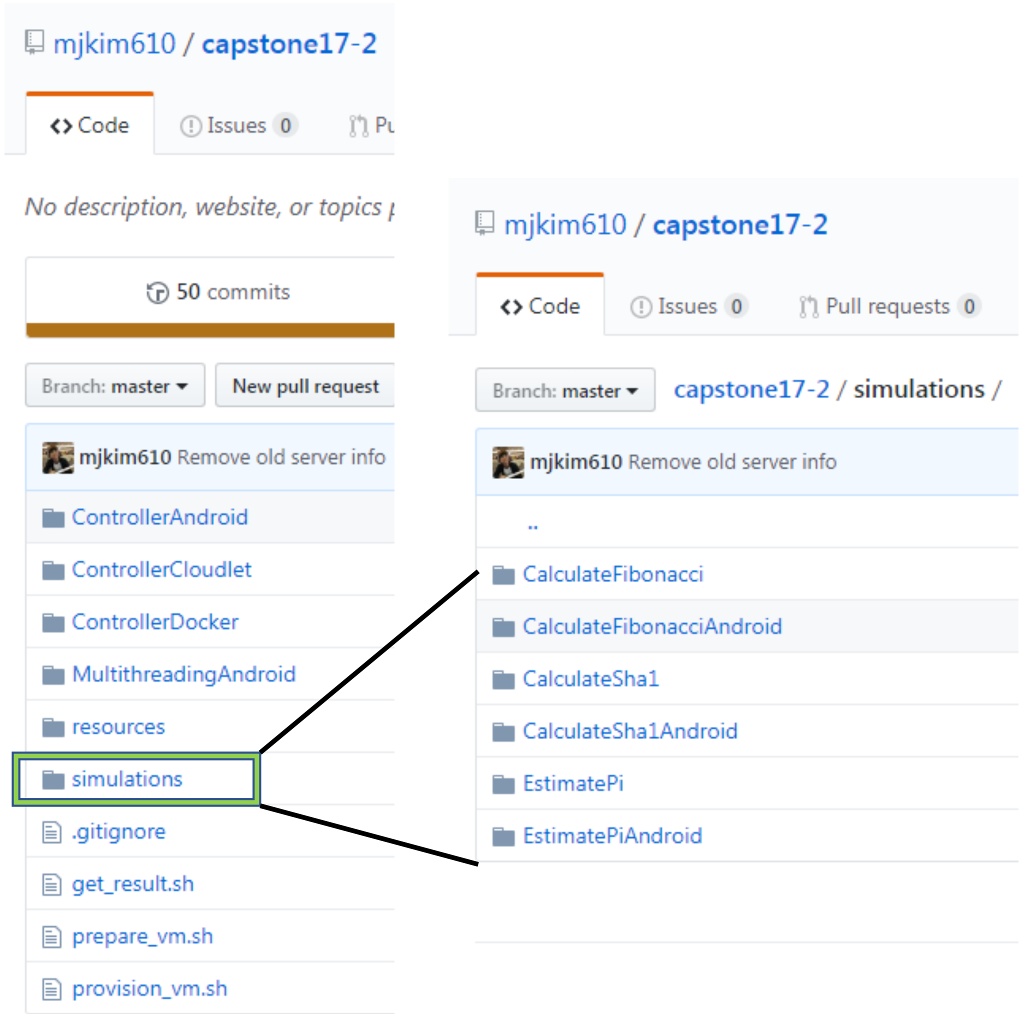
Created by unlimicon
from Noun Project

Created by unlimicon
from Noun Project

Created by unlimicon
from Noun Project

Created by unlimicon
from Noun Project

Created by Kimmi Studio
from Noun Project



mjkim610 / capstone17-2

Code | Issues 0 | Pu

No description, website, or topics

50 commits

Branch: master | New pull request

mjkim610 Remove old server info

- ControllerAndroid
- ControllerCloudlet
- ControllerDocker
- MultithreadingAndroid
- resources
- simulations
- .gitignore
- get_result.sh
- prepare_vm.sh
- provision_vm.sh

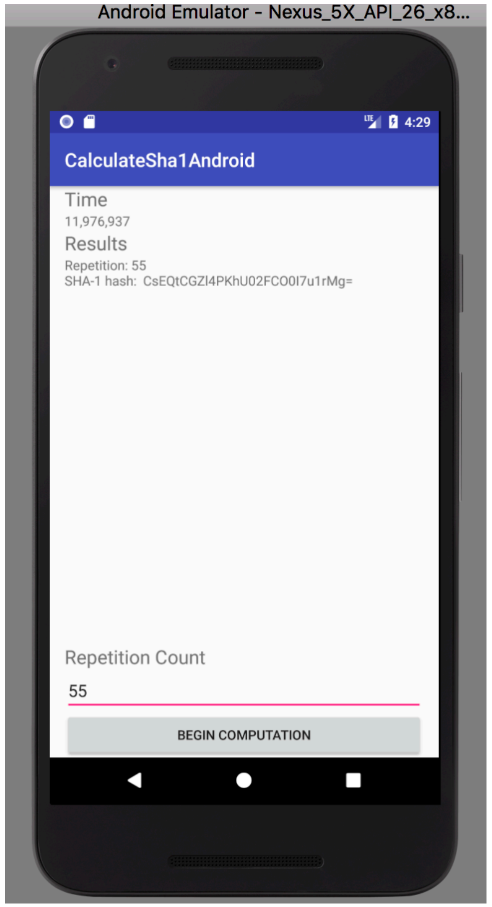
mjkim610 / capstone17-2

Code | Issues 0 | Pull requests 0

Branch: master | capstone17-2 / simulations /

mjkim610 Remove old server info

- ..
- CalculateFibonacci
- CalculateFibonacciAndroid
- CalculateSha1
- CalculateSha1Android
- EstimatePi
- EstimatePiAndroid



Android Emulator - Nexus_5X_API_26_x8...

4:29

CalculateSha1Android

Time
11,976,937
Results
Repetition: 55
SHA-1 hash: CsEQtCGZI4PKhU02FCO0I7u1rMg=

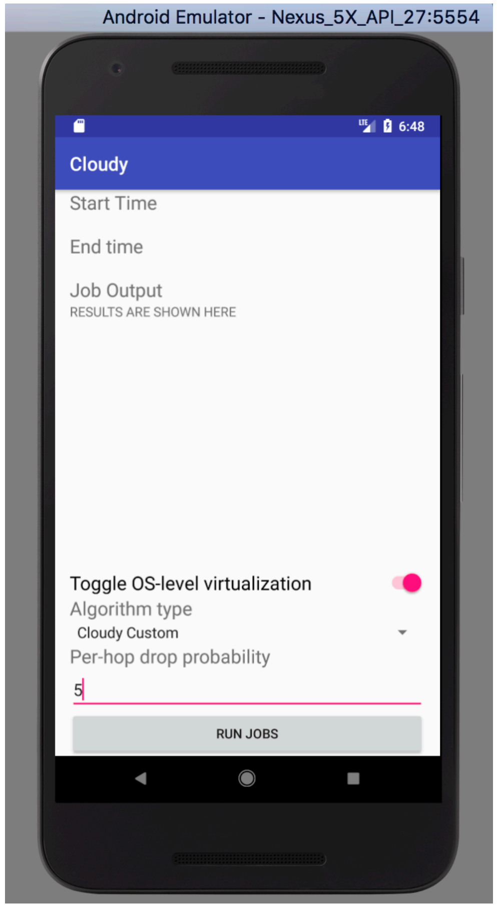
Repetition Count
55

Instance type	CPU clock speed (GHz)	CPU cores	RAM size (GB)	Disk read speed (MB/s)	Disk write speed (MB/s)
t2.2xlarge	2.4	8	16	145	133

Table 4) Characteristics of AWS EC2 instances

CPU clock speed (GHz)	CPU cores	RAM size (GB)	Disk read speed (MB/s)	Disk write speed (MB/s)
3.2	4	16	67.2	54.6

Table 5) Characteristics of edge server


Android Emulator - Nexus_5X_API_27:5554

6:48

Cloudy

Start Time

End time

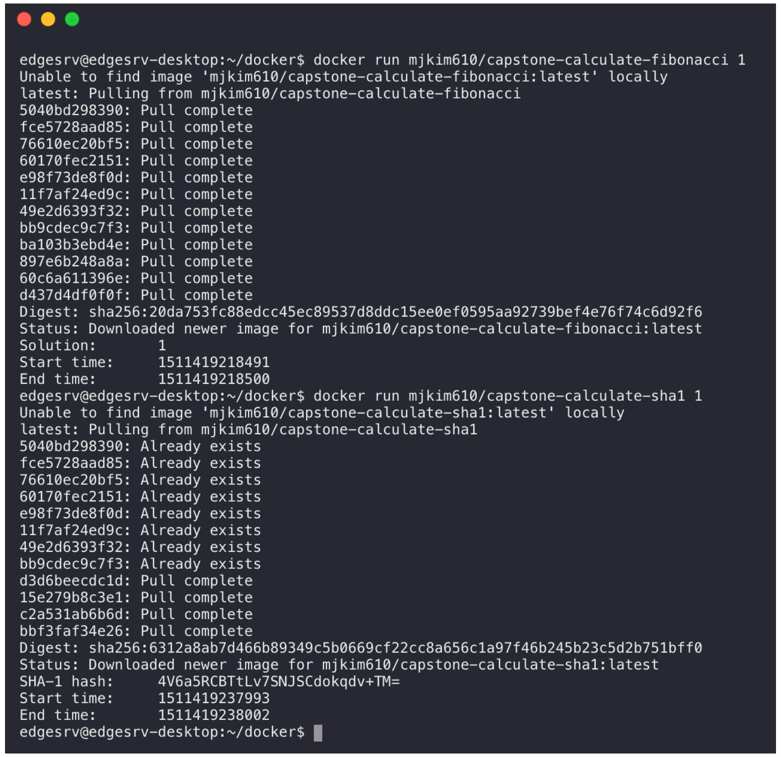
Job Output
RESULTS ARE SHOWN HERE

Toggle OS-level virtualization

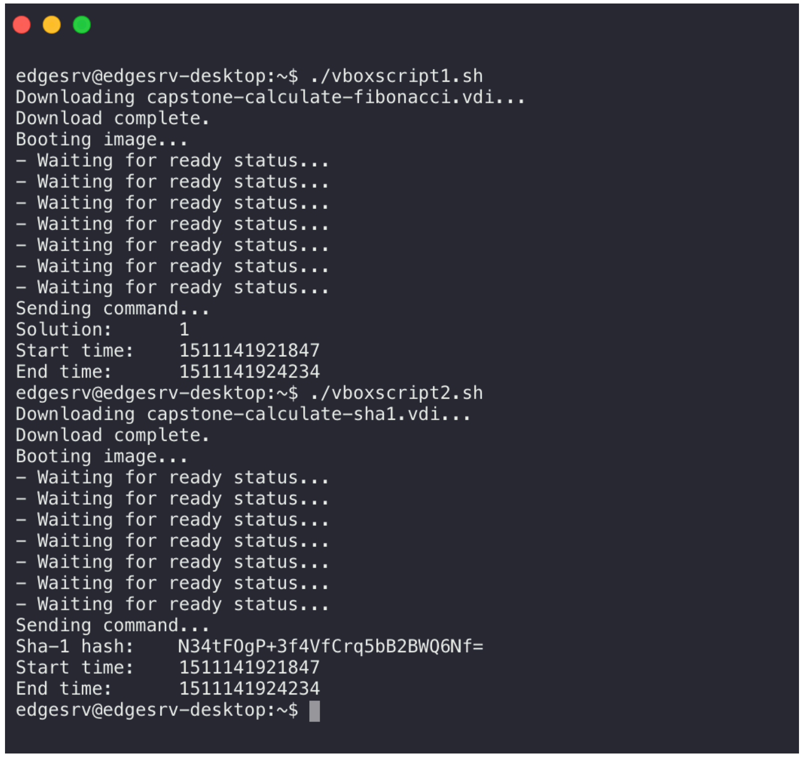
Algorithm type
Cloudy Custom

Per-hop drop probability
5

OS-level virtualization vs. Full virtualization



```
edgesrv@edgesrv-desktop:~/docker$ docker run mjkim610/capstone-calculate-fibonacci 1
Unable to find image 'mjkim610/capstone-calculate-fibonacci:latest' locally
latest: Pulling from mjkim610/capstone-calculate-fibonacci
5040bd298390: Pull complete
fce5728aad85: Pull complete
76610ec20bf5: Pull complete
60170fec2151: Pull complete
e98f73de8f0d: Pull complete
11f7af24ed9c: Pull complete
49e2d6393f32: Pull complete
bb9cdec9c7f3: Pull complete
ba103b3ebd4e: Pull complete
897e6b248a8a: Pull complete
60c6a611396e: Pull complete
d437d4df0f0f: Pull complete
Digest: sha256:20da753fc88edcc45ec89537d8ddc15ee0ef0595aa92739bef4e76f74c6d92f6
Status: Downloaded newer image for mjkim610/capstone-calculate-fibonacci:latest
Solution:       1
Start time:     1511419218491
End time:       1511419218500
edgesrv@edgesrv-desktop:~/docker$ docker run mjkim610/capstone-calculate-sha1 1
Unable to find image 'mjkim610/capstone-calculate-sha1:latest' locally
latest: Pulling from mjkim610/capstone-calculate-sha1
5040bd298390: Already exists
fce5728aad85: Already exists
76610ec20bf5: Already exists
60170fec2151: Already exists
e98f73de8f0d: Already exists
11f7af24ed9c: Already exists
49e2d6393f32: Already exists
bb9cdec9c7f3: Already exists
d3d6beecdc1d: Pull complete
15e279b8c3e1: Pull complete
c2a531ab6b6d: Pull complete
bbf3faf34e26: Pull complete
Digest: sha256:6312a8ab7d466b89349c5b0669cf22cc8a656c1a97f46b245b23c5d2b751bff0
Status: Downloaded newer image for mjkim610/capstone-calculate-sha1:latest
SHA-1 hash:     4V6a5RCBTtLv7SNJSCdokqdv+TM=
Start time:     1511419237993
End time:       1511419238002
edgesrv@edgesrv-desktop:~/docker$
```



```
edgesrv@edgesrv-desktop:~$ ./vboxscript1.sh
Downloading capstone-calculate-fibonacci.vdi...
Download complete.
Booting image...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
Sending command...
Solution:       1
Start time:     1511141921847
End time:       1511141924234
edgesrv@edgesrv-desktop:~$ ./vboxscript2.sh
Downloading capstone-calculate-sha1.vdi...
Download complete.
Booting image...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
- Waiting for ready status...
Sending command...
Sha-1 hash:     N34tFOgP+3f4VfCrq5bB2BWQ6Nf=
Start time:     1511141921847
End time:       1511141924234
edgesrv@edgesrv-desktop:~$
```

OS-level virtualization vs. Full virtualization

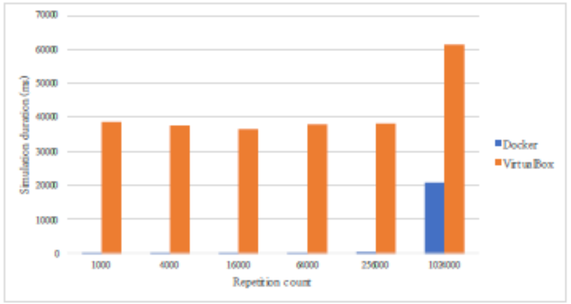


Chart 1) Simulation duration of CalculateFibonacci on Docker and VirtualBox

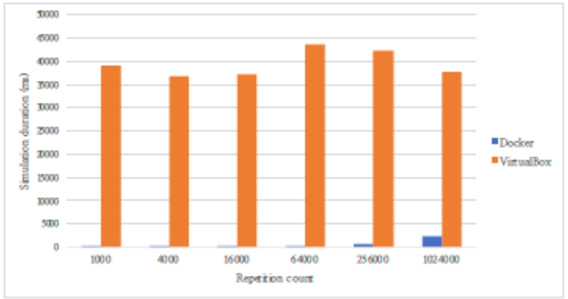


Chart 2) Simulation duration of CalculateSha1 on Docker and VirtualBox

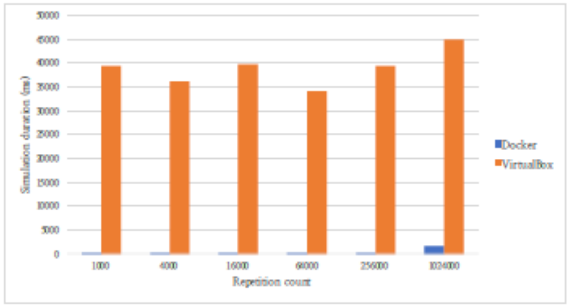
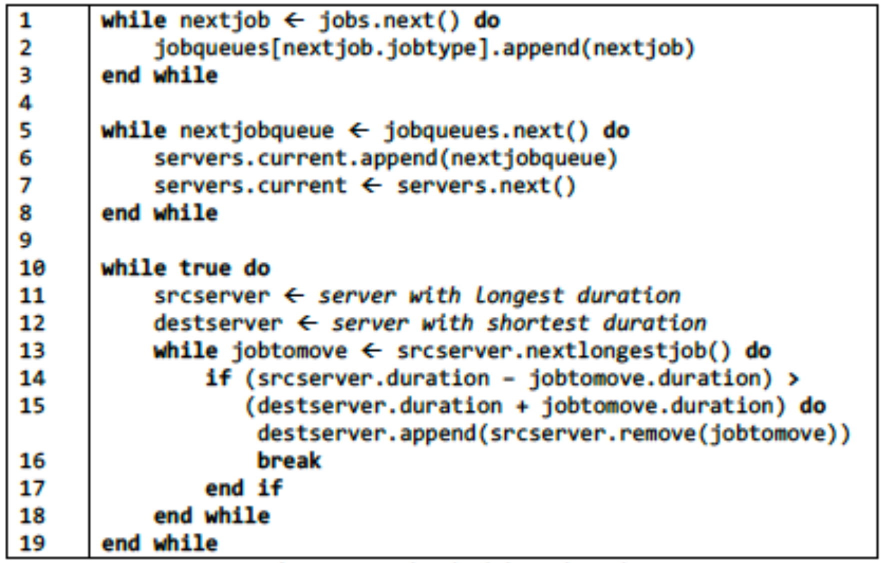


Chart 3) Simulation duration of EstimatePi on Docker and VirtualBox

Custom scheduling algorithm



```
while nextjob ← jobs.next() do
    jobqueues[nextjob.jobtype].append(nextjob)
end while

while nextjobqueue ← jobqueues.next() do
    servers.current.append(nextjobqueue)
    servers.current ← servers.next()
end while

while true do
    srcserver ← server with longest duration
    destserver ← server with shortest duration
    while jobtomove ← srcserver.nextlongestjob() do
        if (srcserver.duration - jobtomove.duration) >
           (destserver.duration + jobtomove.duration) do
            destserver.append(srcserver.remove(jobtomove))
            break
        end if
    end while
end while
```

Fig 1) Custom job scheduling algorithm

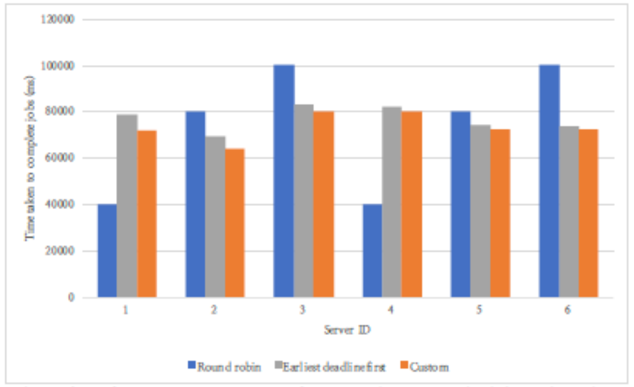


Chart 4) Performance comparison of naïve and custom scheduling algorithms with 3 job types, 6 servers, and 30 jobs

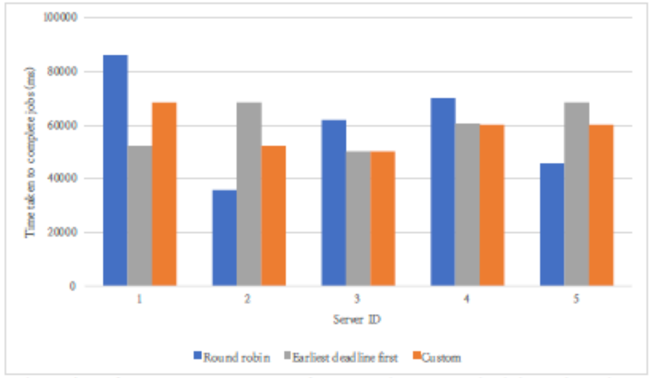


Chart 5) Performance comparison of naïve and custom scheduling algorithms with 4 job types, 5 servers, and 21 jobs

Hop count minimization

Server type	Hop count
Cloudlet – Yonsei University	2
AWS – Seoul	23
AWS – Mumbai	30
AWS – London	25
AWS – Oregon	29
AWS – North Virginia	33

Table 6) Number of hops from mobile device to cloud servers

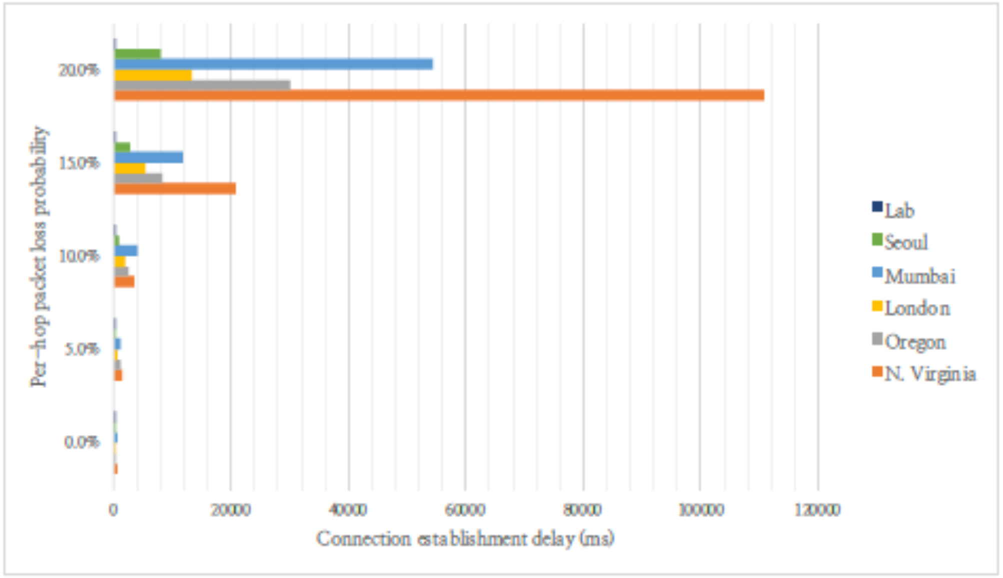


Chart 4) Simulated connection establishment delays between mobile device and servers

4. Result

Real-world scnario:	225,702ms
Theoretical scenario:	265,400ms
Cloudy techniques:	168,043ms
Efficiency improvement: (real-world)	25.55%
Efficiency improvement: (theoretical)	36.68%

Advantages:

- Modular design
- Scalability
- Adaptability
- Better performance at lower load

Disadvantages:

- Requires server components
- Requires additional programming when adding tasks
- Less predictable performance

Q&A